Simple Network Management Protocol Agent for NetX (NetX SNMP)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1054
Revision 5.10

Contents

Chapter 1

Introduction to NetX SNMP

The Simple Network Management Protocol (SNMP) is a protocol designed for managing devices on the internet. SNMP is a protocol that utilizes the connectionless User Datagram Protocol (UDP) services to perform its management function. The NetX SNMP implementation is that of an SNMP Agent. An agent is responsible for responding to SNMP Manager's commands and sending event driven traps.

NetX SNMP Agent Requirements

The NetX SNMP package requires that an IP instance has already been created. In addition, UDP must be enabled on that same IP instance.

The NetX SNMP Agent has several additional requirements. First, it requires complete access to UDP *well-known port 161* for handling all SNMP manager requests. It also requires access to port 162 for sending trap messages to the Manager.

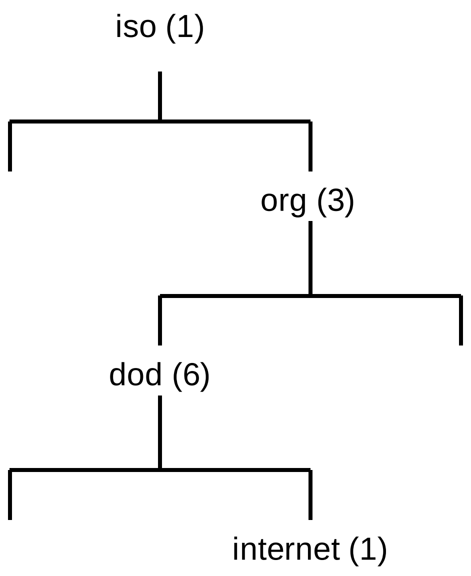
NetX SNMP Constraints

The NetX SNMP protocol implements SNMP Version 1, 2, and 3. The SNMPv3 implementation includes MD5 and SHA authentication, as well as DES encryption. This version of the NetX SNMP Agent does have several constraints, as follows:

1. One SNMP Agent per NetX IP Instance
2. No support for RMON
3. SNMP v3 Informs are not supported

SNMP Object Names

The SNMP protocol is designed to manage devices on the internet. To accomplish this, each SNMP managed device has a set of objects that are defined by the Structure of Management Information (SMI) as defined by RFC 1155. The structure is a hierarchical tree type of structure that looks like the following:



Each node in the tree is an object. The “dod” object in the tree is identified by the notation 1.3.6, while the “internet” object in the tree is identified by the notation 1.3.6.1. All SNMP object names begin with the notation 1.3.6.

An SNMP Manager uses this object notation to specify what object in the device it wishes to get or set. The NetX SNMP Agent interprets such manager requests and provides mechanisms for the application to perform the requested operation.

SNMP Manager Requests

The SNMP has a simple mechanism for managing devices. There is a set of standard SNMP commands that are issued by the SNMP Manager to the SNMP device on port 161. The following shows some of the basic SNMP Manager commands:

SNMP Command	**Meaning**

GET	*Get the specified object*
GETNEXT	*Get the next logical object after the specified object ID*
GETBULK	*Get the multiple logical objects after the specified object ID*
SET	*Set the specified object*

These commands are encoded in the Abstract Syntax Notation One (ASN.1) format and reside in the payload of the UDP packet sent by the SNMP Manager. The NetX SNMP Agent processes the request and then calls the corresponding handling routine specified in the ***nx_snmp_agent_create*** call.

NetX SNMP Agent Traps

The NetX SNMP Agent provides the ability to also alert an SNMP Manager of events asynchronously. This is done via an SNMP trap command. There is a unique API for each version of SNMP for sending traps to an SNMP Manager. By default, the traps are sent to the SNMP Manager on port 162.

The NetX SNMP Agent provides separate security keys for SNMPv3 trap messages. To do so, the SNMP application must create a separate set of keys from those applied to responses to Manager requests. Trap security enables the SNMP Agent to use the same or different passwords for authentication and privacy. For more information on creating security keys, see **NetX SNMP Authentication and Encryption** in the next section.

A list of standard SNMP trap variables is enumerated at the top of *nx_snmp.h*:

```
#define NX_SNMP_TRAP_COLDSTART                0
#define NX_SNMP_TRAP_WARMSTART                1
#define NX_SNMP_TRAP_LINKDOWN                 2
#define NX_SNMP_TRAP_LINKUP                   3
#define NX_SNMP_TRAP_AUTHENTICATE_FAILURE     4
#define NX_SNMP_TRAP_EGPNEIGHBORLOSS          5
#define NX_SNMP_TRAP_ENTERPRISESPECIFIC       6
```

To include these variables in the trap message, the `trap_type` input argument in *nx_snmp_agent_trapv2_send* (SNMPv2) or *nx_snmp_agent_trapv3_send* (SNMPv3) is set to the enumerated value of these variables. An example is shown below for SNMPv2 to notify the SNMP Manager of a cold start event:

```
UINT trap_type = NX_SNMP_TRAP_COLDSTART;

status = nx_snmp_agent_trapv2_send(&my_agent, MIB_IP_ADDRESS,
                                   (UCHAR *)"public", trap_type,
                                   tx_time_get(), NX_NULL);
```

To include proprietary variables in the trap message, the trap_type input argument is set to NX_SNMP_TRAP_CUSTOM and the trap list input argument contains the proprietary data. Note that the trap message will contain as the system up time (1.3.6.1.6.3.1.1.4.1.0). An example is shown below for SNMPv2:

```
NX_SNMP_TRAP_OBJECT trap_list[3];
NX_SNMP_OBJECT_DATA trap_data0;

    /* Load the data into the OBJECT. */
    nx_snmp_object_id_get((void*)"1.3.6.1.4.1.7428.1.3.2.0", &trap_data0);

    /* Update the Trap Object with the object and OID. */
    trap_list[0].nx_snmp_object_string_ptr = (UCHAR *)"1.3.6.1.6.3.1.1.4.0";
    trap_list[0].nx_snmp_object_data  = &trap_data0;

    /* Null terminate the trap list. */
    trap_list[1].nx_snmp_object_string_ptr = NX_NULL;

    status = nx_snmp_agent_trapv2_send(&my_agent, MIB_IP_ADDRESS,
                                       (UCHAR *)"mytrap",
                                       NX_SNMP_TRAP_CUSTOM,
                                        tx_time_get(), trap_list);
```

NetX SNMP Authentication and Encryption

There are two flavors of authentication, namely *basic* and *digest*. Basic authentication is equivalent to a simple plain text *username* authentication found in many protocols. In SNMP basic authentication, the user simply verifies that the supplied username is valid for performing SNMP operations. Basic authentication is the only option for SNMP versions 1 and 2.

The main disadvantage of basic authentication is the username is transmitted in plain text, which makes it easy to steal. The SNMPv3 digest authentication addresses this problem by never transmitting the username in the request. Instead, an algorithm is used to derive a 96-bit key or 'digest' from the username, context engine, and other information. The NetX SNMP Agent supports both the standard MD5 and SHA digest algorithms.

The *snmp_agent_username_process* routine specified in the *nx_snmp_agent_create* call can be used to determine if security will be applied to SNMP Agent messages. It is here the application can reject the supplied username or set up authentication keys and/or privacy keys for the given request. The keys must be created before starting the SNMP agent.

Encryption of SNMPv3 data is available using the DES algorithm. Encryption requires that authentication be enabled (one cannot encrypt data without setting the authentication parameters).

To enable authentication, the SNMP Agent must set its Context Engine ID using the *nx_snmp_agent_context_engine_set* service. The Context Engine ID is used in the creation of the authentication key. Then it must create the keys, and configure the SNMP Agent with those keys.

To create authentication and privacy keys for response messages or for trap messages, the following API are available:

```
UINT  _nx_snmp_agent_md5_key_create(NX_SNMP_AGENT *agent_ptr,
                                    UCHAR *password, NX_SNMP_SECURITY_KEY
                                   *destination_key)

UINT  _nx_snmp_agent_sha_key_create(NX_SNMP_AGENT *agent_ptr,
                                    UCHAR *password, NX_SNMP_SECURITY_KEY
                                   *destination_key)
```

Privacy keys differ from the authentication keys by the input password and choice of digest algorithm.

Next, the SNMP agent must be configured to use these keys. To apply keys for response messages the following API are available:

```
UINT  _nx_snmp_agent_authenticate_key_use(NX_SNMP_AGENT *agent_ptr,
                                          NX_SNMP_SECURITY_KEY *key)

UINT  _nx_snmp_agent_privacy_key_use(NX_SNMP_AGENT *agent_ptr,
                                     NX_SNMP_SECURITY_KEY *key)
```

To apply keys for trap messages the following API are available:

```
UINT  _nx_snmp_agent_auth_trap_key_use(NX_SNMP_AGENT *agent_ptr,
                                       NX_SNMP_SECURITY_KEY *key)

UINT  _nx_snmp_agent_priv_trap_key_use(NX_SNMP_AGENT *agent_ptr,
                                       NX_SNMP_SECURITY_KEY *key)
```

.

NetX SNMP Community Strings

The NetX SNMP Agent supports both public and private community strings. The public string is set with the *nx_snmp_agent_set_public_string* service. The NetX SNMP Agent private string is set using the *nx_snmp_agent_set_private_string* service.

NetX SNMP Username Callback

The NetX SNMP Agent package allows the application to specify (via the ***nx_snmp_agent_create*** call) a username callback routine that is called at the beginning of handling each SNMP Client request.

The callback routine provides the NetX SNMP Agent with the username. If the supplied username is valid or if no username check is necessary for the responding to the request, the username callback should return the value of **NX_SUCCESS**. Otherwise, the routine should return **NX_SNMP_ERROR** to indicate the specified username is invalid.

The format of the application username callback routine is defined below:

```
UINT nx_snmp_agent_username_process(NX_SNMP_AGENT *agent_ptr,
                                    UCHAR *username);
```

The input parameters are defined as follows:

Parameter	**Meaning**
agent_ptr	Pointer to calling SNMP agent
username	Destination for the pointer to the required username

For SNMPv1 and SNMPv2/v2C sessions, the application will want to examine the community string on an incoming SNMP request to determine if the SNMP request has a valid community string. There are several services for the SNMP application to do this.

The SNMP application can inquire if the current SNMP Manager request is a GET (e.g. GET, GETNEXT, GETBULK) or SET type of request using this service:

```
UINT nx_snmp_agent_request_get_type_test(NX_SNMP_AGENT *agent_ptr,
                                         UINT *is_get_type);
```

If the request is a GET type, the application will want to compare the input community string to the SNMP Agent's public string:

```
UINT nx_snmp_agent_public_string_test(NX_SNMP_AGENT *agent_ptr,
                                       UCHAR *username,
                                       UINT *is_public)
```

Similarly if the request is a SET type, the application will want to compare the input community string to the SNMP Agent's private string:

```
UINT nx_snmp_agent_private_string_test(NX_SNMP_AGENT *agent_ptr,
                                        UCHAR *username,
                                        UINT *is_private)
```

The `is_public` and `is_private` return values indicate respectively if the input community string is a valid public or private community string.

The return value of the username callback routine indicates if the username is valid. The value **NX_SUCCESS** is returned if the username is valid, or **NX_SNMP_ERROR** if the username is invalid.

NetX SNMP Agent GET Callback

The application is sets the callback routine responsible for handling GET object requests from the SNMP Manager. The callback retrieves the value of the object specified in the request.

The application GET request callback routine is defined below:

```
UINT nx_snmp_agent_get_process(NX_SNMP_AGENT *agent_ptr,
                                UCHAR *object_requested,
                                NX_SNMP_OBJECT_DATA *object_data);
```

The input parameters are defined as follows:

Parameter	**Meaning**
agent_ptr	Pointer to calling SNMP agent.
object_requested	ASCII string representing the object ID the GET operation is for.
object_data	Data structure to hold the value retrieved by the callback. This can be set with a series of NetX SNMP API's described below.

If the callback function cannot find the requested object, the **NX_SNMP_ERROR_NOSUCHNAME** error code should be returned. If any other error is detected, the **NX_SNMP_ERROR** should be returned.

Note that the "MIB" distributed with the demo programs for the NetX SNMP Agent is intended for testing and development purposes. It is expected that the developer implement their own proprietary MIB for their SNMP application.

NetX SNMP Agent GETNEXT Callback

The application also sets the callback routine responsible for handling GETNEXT object requests from the SNMP Manager. The GETNEXT callback retrieves the value of the next object specified by the request.

The application GETNEXT request callback routine is defined below:

```
UINT nx_snmp_agent_getnext_process(NX_SNMP_AGENT *agent_ptr,
                                   UCHAR *object_requested,
                                   NX_SNMP_OBJECT_DATA *object_data);
```

The input parameters are defined as follows:

Parameter	**Meaning**
agent_ptr	Pointer to calling SNMP agent.
object_requested	ASCII string representing the object ID the GETNEXT operation is for.
object_data	Data structure to hold the value retrieved by the callback. This can be set with a series of NetX SNMP API's described below.

If the callback function cannot find the requested object, the **NX_SNMP_ERROR_NOSUCHNAME** error code should be returned. If any other error is detected, the **NX_SNMP_ERROR** should be returned.

NetX SNMP Agent SET Callback

The application sets the callback routine responsible for handling SET object requests from the SNMP Manager. The SET callback sets the value of the object specified by the request.

The application SET request callback routine is defined below:

```
UINT nx_snmp_agent_set_process(NX_SNMP_AGENT *agent_ptr,
                                UCHAR *object_requested,
                                NX_SNMP_OBJECT_DATA *object_data);
```

The input parameters are defined as follows:

Parameter	**Meaning**
agent_ptr	Pointer to calling SNMP agent.
object_requested	ASCII string representing the object ID the SET operation is for.
object_data	Data structure that contains the new value for the specified object. The actual operation can be done using the NetX SNMP API's described below.

If the callback function cannot find the requested object, the **NX_SNMP_ERROR_NOSUCHNAME** error code should be returned.

If the NetX SNMP host has created private community strings, and the SNMP sender of the SET request does not have the matching private string, it may return an **NX_SNMP_ERROR_NOACCESS** error. If any other error is detected, the **NX_SNMP_ERROR** should be returned.

Changing SNMP Version at Run Time

The SNMP Agent host can change SNMP version to any or all of the three versions at run time using the *nx_snmp_agent_set_version* service. The SNMP Agent is enabled for all three versions when the SNMP Agent is created in *nx_snmp_agent_create*. However, the application can limit that to a subset of all versions. If NX_SNMP_DISABLE_V1 is defined, the SNMP agent drops SNMPV1 messages. Enabling SNMP agent for V1 at run time with *nx_snmp_agent_set_version* will have no effect. The same applies for V2 and V3.

The SNMP Agent can retrieve the SNMP version of the latest SNMP packet received using the *nx_snmp_agent_get_current_version* service.

SNMPv3 Discovery

The SNMP Agent, if enabled for SNMPv3 (e.g. NX_SNMP_DISABLE_V3 must not be defined), will respond to discovery requests from the SNMP Manager. Such a request contains security parameter data with null values for Authoritative Engine ID, user name, boot count and boot time. Authentication is initially disabled. The variable binding list in the request is empty (contains zero items). The SNMP agent responds with the boot data filled in and the variable binding list containing 1 item, *usmStatsUnknownEngineIDs*, which is the count of requests received with an unknown (null) engine ID.

Thereafter the SNMP Manager (or MIB browser) sends its GET, GETNEXT, GETBULK and other SNMP requests to the SNMP Agent with the supplied Engine ID and a username that is known to SNMP Agent. If authentication is enabled, the SNMP Browser and Agent provide an authentication key to ‘prove’ their identity in the *msgAuthoritativeParameters* field of the SNMPv3 header. This message is now authenticated as per the RFC 3414 protocol for SNMPv3 Discovery.

The SNMP agent will respond to a mismatch in zero boot data/boot count in subsequent requests received from the MIB browser with a *NotInTimeWindow* error (if authentication is enabled). The MIB Manager (browser) will then ‘synch’ its boot data and boot count with the SNMP agent boot data in subsequent SNMP requests.

More detailed information on SNMPv3 authentication is available in RFC 3414 “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)”.

NetX SNMP RFCs

NetX SNMP is compliant with RFC1155, RFC1157, RFC1215, RFC1901, RFC1905, RFC1906, RFC1907, RFC1908, RFC2571, RFC2572, RFC2574, RFC2575, RFC 3414 and related RFCs.

.

Chapter 2

Installation and Use of the NetX SNMP Agent

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

Product Distribution

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

nx_snmp.h	Header file for SNMP for NetX
nx_snmp.c	C Source file for SNMP Agent for NetX
nx_md5.c	MD5 digest algorithms
nx_sha.c	SHA digest algorithms
nx_des.c	DES encryption algorithms
nx_snmp.pdf	User Guide for SNMP Agent for NetX
demo_netx_snmp.c	Simple SNMP demonstration
demo_netx_mib2.c	Simple MIB2 demonstration
demo_snmp_helper.h	Header file defining MIB elements

NetX SNMP Agent Installation

In order to use NetX SNMP, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory "*\threadx\arm7\green*" then the *nx_snmp.h*, *nx_snmp.c*, *nx_md5.c, nx_sha.c* and nx*_des.c* files should be copied into this directory.

Using the NetX SNMP Agent

The application must have *nx_snmp.c*, *nx_md5.c, nx_sha.c*, and *nx_des.c* in the build project. The application code must also include *nx_snmp.h* after it includes *nx_api.h* to be able to invoke SNMP services. These files must be compiled in the same manner as other application files and its object form must be linked to the NetX library. This is all that is required to use NetX SNMP.

Note that if **NX_SNMP_NO_SECURITY** is specified in the build process, the *nx_md5.c, nx_sha.c, and nx_des.c* files are not needed.

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

Small Example System

An example of how to use NetX SNMP Agent is described in Figure 1.0 that appears below. In this example, the SNMP include file *nx_snmp.h* is brought in at line 6. The header file that defines the MIB database elements, *demo_snmp_helper.h,* is brought in at line 8. The MIB is defined starting on line 32. Next, the SNMP Agent is created in "*tx_application_define*" at line 129. Note that the SNMP Agent control block "*my_agent*" was defined as a global variable at line 18 previously. SNMP Agent is started at line 229. SNMP object callback definitions for SNMP manager GET, GETNEXT and SET requests, as well as username and MIB update requests, are processed starting at line 250. For this example, no authenticate is performed.

Note that the MIB2 table shown below is simply an example. The application may use a different MIB and include it in separate files, as well as define GET, GETNEXT, or SET processing as per their application requirements.

```
1    /* This is a small demo of the NetX SNMP Agent on the high-performance NetX TCP/IP
2          stack. This demo relies on ThreadX and NetX to show simple SNMP the SNMP
3       GET/GETNEXT/SET requests on MIB-2 objects.  */
4
5    #include  "tx_api.h"
6    #include  "nx_api.h"
7    #include  "nx_snmp.h"
8    #include  "demo_snmp_helper.h"
9
10   #define     DEMO_STACK_SIZE         4096
11   #define     AGENT_PRIMARY_ADDRESS   IP_ADDRESS(192, 2, 2, 66)
12
13   /* Define the ThreadX and NetX object control blocks...  */
14
15   TX_THREAD               thread_0;
16   NX_PACKET_POOL          pool_0;
17   NX_IP                   ip_0;
18   NX_SNMP_AGENT           my_agent;
19
26
27   /* Define authentication and privacy keys.  */
28
29   #ifdef AUTHENTICATION_REQUIRED
30   NX_SNMP_SECURITY_KEY    my_authentication_key;
31   #endif
32
33   #ifdef PRIVACY_REQUIRED
```

```
NX_SNMP_SECURITY_KEY    my_privacy_key;
#endif

/* Define an error counter variable. */
UINT error_counter = 0;

/* This binds a secondary interfaces to the primary IP network interface
   if SNMP is required for required for that interface. */
/* #define  MULTI_HOMED_DEVICE */

/* Define function prototypes.  A generic ram driver is used in this demo.  However
   to properly run an SNMP agent demo, a real driver should be substituted. */

VOID    thread_agent_entry(ULONG thread_input);
VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
UINT    mib2_get_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                            NX_SNMP_OBJECT_DATA *object_data);
UINT    mib2_getnext_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                            NX_SNMP_OBJECT_DATA *object_data);
UINT    mib2_set_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                            NX_SNMP_OBJECT_DATA *object_data);
UINT    mib2_username_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *username);
VOID    mib2_variable_update(NX_IP *ip_ptr, NX_SNMP_AGENT *agent_ptr);


UCHAR context_engine_id[] = {0x80, 0x00, 0x0d, 0xfe, 0x03, 0x00, 0x11, 0x23, 0x23,
                             0x44, 0x55};
UINT  context_engine_size = 11;
UCHAR context_name[] = {0x69, 0x6e, 0x69, 0x74, 0x69, 0x61, 0x6c};
UINT  context_name_size = 7;

/* Define main entry point.  */

int main()
{

    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


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

UCHAR   *pointer;
UINT    status;


    /* Setup the working pointer.  */
    pointer =  (UCHAR *) first_unused_memory;

    status = tx_thread_create(&thread_0, "agent thread", thread_agent_entry, 0,
            pointer, DEMO_STACK_SIZE,
            4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
    if (status != NX_SUCCESS)
    {
         return;
    }

    pointer =  pointer + DEMO_STACK_SIZE;


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

    /* Create packet pool.  */
    status = nx_packet_pool_create(&pool_0, "NetX Packet Pool 0", 2048,
                                    pointer, 20000);

    if (status != NX_SUCCESS)
    {
         return;
    }

    pointer = pointer + 20000;

    /* Create an IP instance.  */
    status = nx_ip_create(&ip_0, "SNMP Agent IP Instance", AGENT_PRIMARY_ADDRESS,
                        0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
                        pointer, 4096, 1);

    if (status != NX_SUCCESS)
    {
        return;
    }

    pointer =  pointer + 4096;

    /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
    nx_arp_enable(&ip_0, (void *) pointer, 1024);
    pointer = pointer + 1024;

    /* Enable UPD processing for IP instance.  */
    nx_udp_enable(&ip_0);

    /* Enable ICMP for ping.  */
    nx_icmp_enable(&ip_0);

    /* Create an SNMP agent instance.  */
    status = nx_snmp_agent_create(&my_agent, "SNMP Agent", &ip_0, pointer, 4096,
                                  &pool_0,
                                  mib2_username_processing, mib2_get_processing,
                                  mib2_getnext_processing,
                                  mib2_set_processing);



    if (status != NX_SUCCESS)
    {
        return;
    }

    pointer =  pointer + 4096;

    status = nx_snmp_agent_context_engine_set(&my_agent, context_engine_id,
                                              context_engine_size);

    if (status != NX_SUCCESS)
    {
        error_counter++;
    }

    return;
}

VOID thread_agent_entry(ULONG thread_input)
{


    /* Allow NetX time to get initialized. */
    tx_thread_sleep(100);

#ifdef AUTHENTICATION_REQUIRED

    /* Create an authentication key.  */
    nx_snmp_agent_md5_key_create(&my_agent, "authpassword", &my_authentication_key);

    /* Use the authentication key.  */
    nx_snmp_agent_authenticate_key_use(&my_agent, &my_authentication_key);
#endif

#ifdef PRIVACY_REQUIRED

    /* Create a privacy key.  */
    nx_snmp_agent_md5_key_create(&my_agent, "privpassword", &my_privacy_key);

    /* Use the privacy key.  */
    nx_snmp_agent_privacy_key_use(&my_agent, &my_privacy_key);
#endif

    /* Start the SNMP instance.  */
    nx_snmp_agent_start(&my_agent);

}

/* Define the application's GET processing routine.  */

UINT    mib2_get_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                             NX_SNMP_OBJECT_DATA *object_data)
{

```

```
UINT    i;
UINT    status;


    printf("SNMP Manager GET Request For:  %s", object_requested);

    /* Loop through the sample MIB to see if we have information for the supplied
       variable.  */
    i =  0;
    status =  NX_SNMP_ERROR;
    while (mib2_mib[i].object_name)
    {

        /* See if we have found the matching entry.  */
        status =  nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);

        /* Was it found?  */
        if (status == NX_SUCCESS)
        {

            /* Yes it was found.  */
            break;
        }

        /* Move to the next index.  */
        i++;
    }

    /* Determine if a not found condition is present.  */
    if (status != NX_SUCCESS)
    {

        printf(" NO SUCH NAME!\n");

        /* The object was not found - return an error.  */
        return(NX_SNMP_ERROR_NOSUCHNAME);
    }

    /* Determine if the entry has a get function.  */
    if (mib2_mib[i].object_get_callback)
    {

        /* Yes, call the get function.  */
        status =  (mib2_mib[i].object_get_callback)(mib2_mib[i].object_value_ptr,
                                                     object_data);
    }
    else
    {

        printf(" NO GET FUNCTION!");

        /* No get function, return no access.  */
        status =  NX_SNMP_ERROR_NOACCESS;
    }

    printf("\n");

    /* Return the status.  */
    return(status);
}


/* Define the application's GETNEXT processing routine.  */

UINT    mib2_getnext_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                                NX_SNMP_OBJECT_DATA *object_data)
{

UINT    i;
UINT    status;


    printf("SNMP Manager GETNEXT Request For:  %s", object_requested);

    /* Loop through the sample MIB to see if we have information for the supplied
       variable.  */
    i =  0;
    status =  NX_SNMP_ERROR;
    while (mib2_mib[i].object_name)
    {

        /* See if we have found the next entry.  */
        status =  nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);

        /* Is the next entry the mib greater?  */
        if (status == NX_SNMP_NEXT_ENTRY)
        {

            /* Yes it was found.  */
            break;
        }

        /* Move to the next index.  */
        i++;
    }

    /* Determine if a not found condition is present.  */
    if (status != NX_SNMP_NEXT_ENTRY)
    {

        printf(" NO SUCH NAME!\n");

        /* The object was not found - return an error.  */
        return(NX_SNMP_ERROR_NOSUCHNAME);
    }


    /* Copy the new name into the object.  */
    nx_snmp_object_copy(mib2_mib[i].object_name, object_requested);

    printf(" Next Name is: %s", object_requested);

    /* Determine if the entry has a get function.  */
    if (mib2_mib[i].object_get_callback)
    {

        /* Yes, call the get function.  */
        status =  (mib2_mib[i].object_get_callback)(mib2_mib[i].object_value_ptr,
                                                    object_data);

        /* Determine if the object data indicates an end-of-mib condition.  */
        if (object_data -> nx_snmp_object_data_type == NX_SNMP_END_OF_MIB_VIEW)
        {

            /* Copy the name supplied in the mib table.  */
            nx_snmp_object_copy(mib2_mib[i].object_value_ptr, object_requested);
        }
    }
    else
    {

        printf(" NO GET FUNCTION!");

        /* No get function, return no access.  */
        status =  NX_SNMP_ERROR_NOACCESS;
    }

    printf("\n");

    /* Return the status.  */
    return(status);
}


/* Define the application's SET processing routine.  */

UINT    mib2_set_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
                            NX_SNMP_OBJECT_DATA *object_data)
{

UINT    i;
UINT    status;


    printf("SNMP Manager SET Request For:  %s", object_requested);

    /* Loop through the sample MIB to see if we have information for the supplied
variable.  */
    i =  0;
    status =  NX_SNMP_ERROR;
    while (mib2_mib[i].object_name)
    {

        /* See if we have found the matching entry.  */
        status =  nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);

        /* Was it found?  */
        if (status == NX_SUCCESS)
        {

            /* Yes it was found.  */
            break;
        }

        /* Move to the next index.  */
        i++;
    }

    /* Determine if a not found condition is present.  */
    if (status != NX_SUCCESS)
    {

        printf(" NO SUCH NAME!\n");

        /* The object was not found - return an error.  */
        return(NX_SNMP_ERROR_NOSUCHNAME);
    }


    /* Determine if the entry has a set function.  */
    if (mib2_mib[i].object_set_callback)
    {

        /* Yes, call the set function.  */
        status =  (mib2_mib[i].object_set_callback)(mib2_mib[i].object_value_ptr,
                                                     object_data);
    }
    else
    {

        printf(" NO SET FUNCTION!");

        /* No get function, return no access.  */
        status =  NX_SNMP_ERROR_NOACCESS;
    }

    printf("\n");

    /* Return the status.  */
    return(status);
}


/* Define the application's authentication routine.  */

UINT  mib2_username_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *username)
{

    printf("Username is:  %s\n", username);

    /* Update MIB-2 objects. In this example, it is only the SNMP objects.  */
    mib2_variable_update(&ip_0, &my_agent);

    /* No authentication is done, just return success!  */
    return(NX_SUCCESS);
}


/* Define the application's update routine.  */

VOID  mib2_variable_update(NX_IP *ip_ptr, NX_SNMP_AGENT *agent_ptr)
{

    /* Update the snmp parameters.  */
    snmpInPkts =                agent_ptr -> nx_snmp_agent_packets_received;
    snmpOutPkts =               agent_ptr -> nx_snmp_agent_packets_sent;
    snmpInBadVersions =         agent_ptr -> nx_snmp_agent_invalid_version;
    snmpInBadCommunityNames =   agent_ptr -> nx_snmp_agent_authentication_errors;
    snmpInBadCommunityUsers =   agent_ptr -> nx_snmp_agent_username_errors;
    snmpInASNParseErrs =        agent_ptr -> nx_snmp_agent_internal_errors;
    snmpInTotalReqVars =        agent_ptr -> nx_snmp_agent_total_get_variables;
    snmpInTotalSetVars =        agent_ptr -> nx_snmp_agent_total_set_variables;
    snmpInGetRequests =         agent_ptr -> nx_snmp_agent_get_requests;
    snmpInGetNexts =            agent_ptr -> nx_snmp_agent_getnext_requests;
    snmpInSetRequests =         agent_ptr -> nx_snmp_agent_set_requests;
    snmpOutTooBigs =            agent_ptr -> nx_snmp_agent_too_big_errors;
    snmpOutNoSuchNames =        agent_ptr -> nx_snmp_agent_no_such_name_errors;
    snmpOutBadValues =          agent_ptr -> nx_snmp_agent_bad_value_errors;
    snmpOutGenErrs =            agent_ptr -> nx_snmp_agent_general_errors;
    snmpOutTraps =              agent_ptr -> nx_snmp_agent_traps_sent;
}

```

Figure 1.0 Example of SNMP Agent use with NetX

Configuration Options

There are several configuration options for building SNMP for NetX. Following is a list of all options, where each is described in detail:

Define	Meaning
NX_SNMP_AGENT_PRIORITY	The priority of the SNMP AGENT thread. By default, this value is defined as 16 to specify priority 16.
NX_SNMP_TYPE_OF_SERVICE	Type of service required for the SNMP UDP responses. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service. This define can be set by the application prior to inclusion of *nx_snmp.h*.
NX_SNMP_FRAGMENT_OPTION	Fragment enable for SNMP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable SNMP UDP fragmenting. This define can be set by the application prior to inclusion of *nx_snmp.h*.
NX_SNMP_TIME_TO_LIVE	Specifies the time to live before it expires. The default value is set to 0x80, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_AGENT_TIMEOUT	Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 100, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_MAX_OCTET_STRING	Specifies the maximum number of bytes allowed in an octet string in the SNMP Agent.

	The default value is set to 255, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_MAX_CONTEXT_STRING	Specifies the maximum number of bytes for a context engine string in the SNMP Agent. The default value is set to 32, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_MAX_USER_NAME	Specifies the maximum number of bytes in a username (including community strings). The default value is set to 64, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_MAX_SECURITY_KEY	Specifies the number of bytes allowed in a security key string. The default value is set to 64, but can be redefined prior to nclusion of *nx_snmp.h.*
NX_SNMP_PACKET_SIZE	Specifies the minimum size of the packets in the pool specified at SNMP Agent creation. The minimum size is needed to ensure the complete SNMP payload can be contained in one packet. The default value is set to 560, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_AGENT_PORT	Specifies the UDP port to field SNMP Manager requests on. The default port is UDP port 161, but can be redefined prior to inclusion of *nx_snmp.h.*
NX_SNMP_MANAGER_TRAP_PORT	Specifies the UDP port to send SNMP Agent trap requests to. The default port is UDP port 162, but can be redefined prior to

	inclusion of *nx_snmp.h.*
NX_SNMP_MAX_TRAP_NAME	Specifies the size of the array to hold the username sent with trap messages. The default value is 64.
NX_SNMP_MAX_TRAP_KEY	Specifies the size of the authentication and privacy keys for trap messages. The default value is 64.
NX_SNMP_TIME_INTERVAL	This determines the sleep interval in timer ticks taken by the SNMP thread task between processing received SNMP packets. The default value is 100. During this sleep interval the host application has access to SNMP API services.
NX_SNMP_DISABLE_V1	Defined, this removes all the SNMP Version 1 processing in *nx_snmp.c*. By default this is not defined.
NX_SNMP_DISABLE_V2	Defined, this removes all the SNMP Version 2 processing in *nx_snmp.c*. By default this is not defined.
NX_SNMP_DISABLE_V3	Defined, this removes all the SNMPv3 processing in *nx_snmp.c*. By default this is not defined.

Chapter 3

Description of SNMP Agent Services

This chapter contains a description of all NetX SNMP Agent 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_snmp_agent_auth_trap_key_use
Specify authentication key (SNMP v3 only) for trap messages

nx_snmp_agent_authenticate_key_use
Specify authentication key (SNMP v3 only) for response messages

nx_snmp_agent_community_get
Retrieve community name

nx_snmp_agent_context_engine_set
Set context engine (SNMP v3 only)

nx_snmp_agent_context_name_set
Set context name (SNMP v3 only)

nx_snmp_agent_create
Create SNMP agent

nx_snmp_agent_current_version_set
Get the SNMP version of received packet

nx_snmp_agent_request_get_type_test
Indicate if last SNMP request is GET or SET type

nx_snmp_agent_private_string_test
Determine if string matches agent private string

nx_snmp_agent_public_string_test

Determine if string matches agent public string

nx_snmp_agent_set_interface
Set network interface for SNMP messaging

nx_snmp_agent_private_string_set
Set the SNMP agent private community string

nx_snmp_agent_public_string_set
Set the SNMP agent public community string

nx_snmp_agent_version_set
Set the SNMP agent status for all SNMP versions

nx_snmp_agent_delete
Delete SNMP agent

nx_snmp_agent_md5_key_create
Create md5 key (SNMP v3 only)

nx_snmp_agent_ priv_trap_key_use
Specify encryption key (SNMP v3 only) for trap messages

nx_snmp_agent_privacy_key_use
Specify encryption key (SNMP v3 only) for response messages

nx_snmp_agent_sha_key_create
Create sha key (SNMP v3 only)

nx_snmp_agent_start
Start SNMP agent

nx_snmp_agent_stop
Stop SNMP agent

nx_snmp_agent_trap_send
Send SNMP v1 trap

nx_snmp_agent_trapv2_send
Send SNMP v2 trap

nx_snmp_agent_trapv2_send_oid
Send SNMP v2 trap specifying the OID

nx_snmp_agent_trapv3_send
Send SNMP v3 trap

nx_snmp_agent_trapv3_send_oid
Send SNMP v2 trap specifying the OID

nx_snmp_agent_v3_context_boots_set
Set the number of reboots

nx_snmp_object_compare
Compare two objects

nx_snmp_object_copy
Copy an object

nx_snmp_object_counter_get
Get counter object

nx_snmp_object_counter_set
Set counter object

nx_snmp_object_counter64_get
Get 64-bit counter object

nx_snmp_object_counter64_set
Set 64-bit counter object

nx_snmp_object_end_of_mib
Set end-of-mib value

nx_snmp_object_gauge_get
Get gauge object

nx_snmp_object_gauge_set
Set gauge object

nx_snmp_object_id_get
Get object id

nx_snmp_object_id_set
Set object id

nx_snmp_object_integer_get
Get integer object

nx_snmp_object_integer_set

Set integer object

nx_snmp_object_ip_address_get
Get IP address object

nx_snmp_object_ip_address_set
Set IP address object

nx_snmp_object_no_instance
Set no-instance value

nx_snmp_object_not_found
Set not-found value

nx_snmp_object_octet_string_get
Get octet string object

nx_snmp_object_octet_string_set
Set octet string object

nx_snmp_object_string_get
Get ASCII string object

nx_snmp_object_string_set
Set ASCII string object

nx_snmp_object_timetics_get
Get timetics object

nx_snmp_object_timetics_set
Set timetics object

nx_snmp_agent_auth_trap_key_use

Specify authentication key for trap messages

Prototype

```
UINT nx_snmp_agent_auth_trap_key_use(NX_SNMP_AGENT *agent_ptr,
                                     NX_SNMP_SECURITY_KEY *key);
```

Description

This service specifies a previously created key to be used for setting authentication parameters in the SNMPv3 security header in trap messages. Supplying a NX_NULL value for the key disables authentication.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
key	Pointer to a previously created MD5 or SHA key.

Return Values

NX_SUCCESS	(0x00)	Successful authentication key set.
NX_NOT_ENABLED	(0x14)	SNMP Security disabled
NX_PTR_ERROR	(0x07)	Invalid SNMP Agent pointer.

Allowed From

Initialization, Threads

Example

```
/* Use previously created “my_key” for SNMP v3 trap message authentication.  */
status =  nx_snmp_agent_auth_trap_key_use(&my_agent, &my_key);


/* If status is NX_SUCCESS the SNMP Agent will use “my_key” for
   for authentication parameters in trap messages.  */
```

nx_snmp_agent_authenticate_key_use

Specify authentication key for response messages

Prototype

```
UINT nx_snmp_agent_authenticate_key_use(NX_SNMP_AGENT *agent_ptr,
                                        NX_SNMP_SECURITY_KEY *key);
```

Description

This service specifies a previously created key to be used for authentication parameters in the SNMPv3 security parameter for all requests made after it is set. Supplying a NX_NULL value for the key disables authentication.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
key	Pointer to a previously created MD5 or SHA key.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP key set.
NX_NOT_ENABLED	(0x14)	SNMP Security disabled
NX_PTR_ERROR	(0x07)	Invalid SNMP Agent pointer.

Allowed From

Initialization, Threads

Example

```
/* Use previously created “my_key” for SNMP v3 authentication.  */
status =  nx_snmp_agent_authenticate_key_use(&my_agent, &my_key);


/* If status is NX_SUCCESS the SNMP Agent will use “my_key” for
   for setting the authentication parameters of SNMPv3 requests.  */
```

nx_snmp_agent_community_get

Retrieve community name

Prototype

```
UINT nx_snmp_agent_community_get(NX_SNMP_AGENT *agent_ptr,
                                 UCHAR **community_string_ptr);
```

Description

This service retrieves the community name from the most recent SNMP request received by the SNMP Agent.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string_ptr	Pointer to a string pointer to return the SNMP Agent community string.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP community get.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
UCHAR *string_ptr;

/* Pickup the community string pointer for my_agent.  */
status =  nx_snmp_agent_community_get(&my_agent, &string_ptr);


/* If status is NX_SUCCESS the pointer “string_ptr” points to the
   last community name supplied to the SNMP agent.  */
```

nx_snmp_agent_request_get_type_test

Indicate if a request is GET type

Prototype

```
UINT nx_snmp_agent_request_get_type_test(NX_SNMP_AGENT *agent_ptr,
                                         UINT *is_get_type);
```

Description

This service indicates if the most recent request from the SNMP Manager is a GET (GET, GETNEXT, or GETBULK) or SET type. It is intended for use with the username callback where the SNMPv1 or SNMPv2 application will want to compare the received community string to the SNMP Agent public string if the request is a GET type, or to the SNMP Agent private string if the request is a SET type.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
is_get_type	Pointer to request type status: NX_TRUE if GET type NX_FALSE if SET type

Return Values

NX_SUCCESS	(0x00)	Successfully returned type
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
UINT is_get_type;

/* Determine if the current SNMP request is a GET or SET type.  */
status =  nx_snmp_agent_request_get_type_test(&my_agent, &is_get_type);


/* If status is NX_SUCCESS, is_get_type will indicate the request type.  */
```

nx_snmp_agent_context_engine_set

Set context engine (SNMP v3 only)

Prototype

```
UINT nx_snmp_agent_context_engine_set(NX_SNMP_AGENT *agent_ptr,
                                       UCHAR *context_engine,
                                       UINT context_engine_size);
```

Description

This service sets the context engine of the SNMP Agent. It is only applicable for SNMPv3 processing. This should be called before creating keys if the application is using authentication and encryption, since the context engine ID is used in the key creation process. If not, NetX SNMP provides a default context engine id at the top of *nx_snmp.c:*

```
 UCHAR _nx_snmp_default_context_engine[NX_SNMP_MAX_CONTEXT_STRING] =
                {0x80, 0x00, 0x03, 0x10, 0x01, 0xc0, 0xa8, 0x64, 0xaf};
```

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
context_engine	Pointer to the context engine string.
context_engine_size	Size of context engine string. Note that the maximum number of bytes in a context engine is defined by NX_SNMP_MAX_CONTEXT_STRING.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP context engine set.
NX_NOT_ENABLED	(0x14)	SNMPv3 is not enabled
NX_SNMP_ERROR	(0x100)	Context engine size error.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
UCHAR my_engine[] = {0x80, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};

/* Set the context engine for my_agent.  */
status =  nx_snmp_agent_context_engine_set(&my_agent, my_engine, 9);


/* If status is NX_SUCCESS the context engine has been set.  */
```

nx_snmp_agent_context_name_set

Set context name (SNMP v3 only)

Prototype

```
UINT nx_snmp_agent_context_name_set(NX_SNMP_AGENT *agent_ptr,
                                    UCHAR *context_name,
                                    UINT context_name_size);
```

Description

This service sets the context name of the SNMP Agent. It is only applicable for SNMPv3 processing. If not called, NetX SNMP Agent will leave the context name blank.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
context_name	Pointer to the context name string.
context_name_size	Size of context name string. Note that the maximum number of bytes in a context name is defined by NX_SNMP_MAX_CONTEXT_STRING.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP context name set.
NX_SNMP_ERROR	(0x100)	Context name size error.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Set the context name for my_agent.  */
status =  nx_snmp_agent_context_name_set(&my_agent, “my_context_name”, 15);


/* If status is NX_SUCCESS the context name has been set.  */
```

nx_snmp_agent_create

Create SNMP agent

Prototype

```
UINT nx_snmp_agent_create(NX_SNMP_AGENT *agent_ptr,
      CHAR *snmp_agent_name, NX_IP *ip_ptr, VOID *stack_ptr,
      ULONG stack_size, NX_PACKET_POOL *pool_ptr,
      UINT (*snmp_agent_username_process)(struct NX_SNMP_AGENT_STRUCT
                              *agent_ptr, UCHAR *username),
      UINT (*snmp_agent_get_process)(struct NX_SNMP_AGENT_STRUCT
                  *agent_ptr, UCHAR *object_requested,
                  NX_SNMP_OBJECT_DATA *object_data),
      UINT (*snmp_agent_getnext_process)(struct NX_SNMP_AGENT_STRUCT
                  *agent_ptr, UCHAR *object_requested,
                  NX_SNMP_OBJECT_DATA *object_data),
      UINT (*snmp_agent_set_process)(struct NX_SNMP_AGENT_STRUCT
                  *agent_ptr, UCHAR *object_requested,
                  NX_SNMP_OBJECT_DATA *object_data));
```

Description

This service creates a SNMP Agent on the specified IP instance.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
snmp_agent_name	Pointer to the SNMP Agent name string.
ip_ptr	Pointer to IP instance.
stack_ptr	Pointer to SNMP Agent thread stack pointer.
stack_size	Stack size in bytes.
pool_ptr	Pointer the default packet pool for this SNMP Agent.
snmp_agent_username_process	Function pointer to application's username handling routine.
snmp_agent_get_process	Function pointer to application's GET request handling routine.

snmp_agent_getnext_process

Function pointer to application’s GETNEXT request handling routine.

snmp_agent_set_process

Function pointer to application’s SET request handling routine.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP Agent create.
NX_SNMP_ERROR	(0x100)	SNMP Agent create error.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
NX_SNMP_AGENT my_agent;

/* Create the SNMP Agent “my_agent.”  */
status =  nx_snmp_agent_create(&my_agent, "My SNMP Agent", &ip_0, stack_start_ptr,
                4096, &pool_0, my_username_processing, my_get_processing,
                my_getnext_processing, my_set_processing);


/* If status is NX_SUCCESS the SNMP Agent “my_agent” has been created.  */
```

nx_snmp_agent_current_version_get

Get the SNMP packet version

Prototype

```
UINT nx_snmp_agent_current_version_get(NX_SNMP_AGENT *agent_ptr,
                                       UINT *version);
```

Description

This service retrieves the SNMP version parsed from the most recent SNMP packet received.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
version	Pointer to the SNMP version parsed from received SNMP packet

Return Values

NX_SUCCESS	(0x00)	Successful SNMP version get
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
UINT          snmp_version;
NX_SNMP_AGENT my_agent;

/* Get the version of the last received SNMP packet  */
status =  nx_snmp_agent_current_version_get (&my_agent, &snmp_version);


/* If status is NX_SUCCESS, snmp_version contains the received packet SNMP
version.  */
```

nx_snmp_agent_private_string_test

Verify private string matches Agent private string

Prototype

```
UINT nx_snmp_agent_private_string_test(NX_SNMP_AGENT *agent_ptr,
                                       UCHAR *community_string,
                                       UINT *is_private);
```

Description

This service compares the null terminated input community string parsed from the received SNMP request with the SNMP agent private string and indicates if they match.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string	Pointer to string to compare
is_private	Pointer to result of comparison NX_TRUE - string matches NX_FALSE - string does not match

Return Values

NX_SUCCESS	(0x00)	Successful comparison
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
UINT           is_private;
UCHAR          *community_string_ptr;
NX_SNMP_AGENT  my_agent;

/* Determine if the community string extracted matches the agent private string */
status =  nx_snmp_agent_private_string_test(&my_agent, community_string_ptr,
                                            &is_private);


/* If status is NX_SUCCESS, is_private will indicate if there is a match. If is_
private is NX_TRUE they match.  */
```

nx_snmp_agent_public_string_test

Verify public string matches Agent public string

Prototype

```
UINT nx_snmp_agent_public_string_test(NX_SNMP_AGENT *agent_ptr,
                                       UCHAR *community_string,
                                       UINT *is_public);
```

Description

This service compares a null terminated input community string parsed from the received SNMP packet with the SNMP agent public string and indicates if they match.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string	Pointer to string to compare
is_public	Pointer to result of comparison NX_TRUE - string matches NX_FALSE - string does not match

Return Values

NX_SUCCESS	(0x00)	Successful comparison
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
UINT           is_public;
UCHAR          *community_string_ptr;
NX_SNMP_AGENT  my_agent;


/* Determine if the community string extracted matches the agent public string */
status =  nx_snmp_agent_public_string_test(&my_agent, community_string_ptr,
                                           &is_public);


/* If status is NX_SUCCESS, is_public will indicate if there is a match. If
is_public is NX_TRUE they match.  */
```

nx_snmp_agent_version_set

Set the SNMP agent enabled status for all SNMP versions

Prototype

```
UINT nx_snmp_agent_version_set(NX_SNMP_AGENT *agent_ptr,
                               UINT enabled_v1, UINT enable_v2,
                               UINT enable_v3);
```

Description

This service sets the status (enabled/disabled) in the SNMP agent for each of the SNMP versions, V1, V2 and V3. Note that the user configurable options, NX_SNMP_DISABLE_V1, NX_SNMP_DISABLE_V2, and NX_SNMP_DISABLE_V3, if set will override these run time settings. If not disabled, the SNMP agent version enable status is determined by an internal runtime variable. By default, the SNMP agent is enabled for all three versions.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
enabled_v1	Sets enabled status for SNMP V1 to on/off.
enabled_v2	Sets enabled status for SNMP V2 to on/off.
enabled_v3	Sets enabled status for SNMP V3 to on/off.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP version set
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
UINT          v1_on = NX_TRUE;
UINT          v2_on = NX_TRUE;
UINT          v3_on = NX_FALSE;
```

```
NX_SNMP_AGENT my_agent;

/* Get the version of the last received SNMP packet  */
status =  nx_snmp_agent_version_set(&my_agent, v1_on, v2_on, v3_on);


/* If status is NX_SUCCESS, my_agent is enabled only for V1 and V2 assuming
NX_SNMP_DISABLE_V1 and NX_SNMP_DISABLE_V2 are not defined. */
```

nx_snmp_agent_private_string_set

Set the SNMP agent private string

Prototype

```
UINT nx_snmp_agent_private_string_set(NX_SNMP_AGENT *agent_ptr,
                                      UCHAR *community_string);
```

Description

This service sets the SNMP agent private community string with the input null terminated string. The default value is NULL (no private string set, such that any SNMP packet received with a “private” community string will not be accepted by the SNMP agent for read/write access. The input string must be less than or equal to the user configurable NX_SNMP_MAX_USER_NAME-1 (to allow room for null termination) size.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string	Pointer to the private string to assign

Return Values

NX_SUCCESS	(0x00)	Successfully set private string
NX_SNMP_ERROR_TOOBIG	(0x01)	String size too large
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
NX_SNMP_AGENT my_agent;

/* Set the SNMP agent’s private community string */
status =  nx_snmp_agent_private_string_set(&my_agent, “private”));


/* If status is NX_SUCCESS, the SNMP agent private string is set.  */
```

nx_snmp_agent_public_string_set

Set the SNMP agent public string

Prototype

```
UINT nx_snmp_agent_public_string_set(NX_SNMP_AGENT *agent_ptr,
                                      UCHAR *community_string);
```

Description

This service sets the SNMP agent public community string with the input null terminated string. The input community string must be less than or equal to the user configurable NX_SNMP_MAX_USER_NAME-1 (to allow room for null termination) size.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string	Pointer to the public string to assign

Return Values

NX_SUCCESS	(0x00)	Successfully set public string
NX_SNMP_ERROR_TOOBIG	(0x01)	String size too large
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
NX_SNMP_AGENT my_agent;

/* Set the SNMP agent's public string. */
nx_snmp_agent_public_string_set(&my_agent, "my_public"));


/* If status is NX_SUCCESS, the SNMP agent public string is set.  */
```

nx_snmp_agent_delete

Delete SNMP agent

Prototype

```
UINT nx_snmp_agent_delete(NX_SNMP_AGENT *agent_ptr);
```

Description

This service deletes a previously created SNMP Agent.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP Agent delete.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Delete the SNMP Agent “my_agent.”  */
status =  nx_snmp_agent_delete(&my_agent);


/* If status is NX_SUCCESS the SNMP Agent “my_agent” has been deleted.  */
```

nx_snmp_agent_set_interface

Set the SNMP agent network interface

Prototype

```
UINT nx_snmp_agent_set_interface(NX_SNMP_AGENT *agent_ptr,
                                 UINT if_index);
```

Description

This service sets the SNMP network interface for the SNMP Agent as specified by the input interface index. This is only useful for SNMP host applications with NetX 5.6 or higher which support multiple network interfaces. The default value if not specified by the host is zero, for the primary interface.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
If_index	Index specifying the SNMP interface.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP interface set.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Set the SNMP Agent “my_agent” to the secondary interface.  */
if_index = 1;
status =  nx_snmp_agent_set_interface(&my_agent, if_index);


/* If status is NX_SUCCESS the SNMP Agent “my_agent” has its interface set.  */
```

nx_snmp_agent_md5_key_create

Create md5 key (SNMP v3 only)

Prototype

```
UINT nx_snmp_agent_md5_key_create(NX_SNMP_AGENT *agent_ptr,
                                  UCHAR *password,
                                  NX_SNMP_SECURITY_KEY
                                       *destination_key);
```

Description

This service creates a MD5 key that can be used for authentication and encryption.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
password	Pointer to password string.
destination_key	Pointer to SNMP key data structure.

Return Values

NX_SUCCESS	(0x00)	Successful key create.
NX_NOT_ENABLED	(0x14)	Security not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
NX_SNMP_SECURITY_KEY my_key;

/* Create the MD5 key for “my_agent.”   */
status =  nx_snmp_agent_md5_key_create(&my_agent, “authpw”, &my_key);


/* If status is NX_SUCCESS the key for the password “authpw” has been created.  */
```

nx_snmp_agent_priv_trap_key_use

Specify encryption key for trap messages

Prototype

```
UINT nx_snmp_agent_priv_trap_key_use(NX_SNMP_AGENT *agent_ptr,
                                     NX_SNMP_SECURITY_KEY *key);
```

Description

This service specifies that a previously created privacy key is to be used for encryption and decryption of SNMPv3 trap messages.

Note that an authentication key must also be created. SNMPv3 encryption also requires authentication.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
key	Pointer to previously create key.

Return Values

NX_SUCCESS	(0x00)	Successful privacy key set.
NX_NOT_ENABLED	(0x14)	Security not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Use the “my_privacy_key” for the SNMP Agent “my_agent” trap messages.   */
status =  nx_snmp_agent_priv_trap_key_use(&my_agent, &my_privacy_key);


/* If status is NX_SUCCESS the privacy key has been set.  */
```

nx_snmp_agent_privacy_key_use

Specify encryption key for response messages

Prototype

```
UINT nx_snmp_agent_privacy_key_use(NX_SNMP_AGENT *agent_ptr,
                                   NX_SNMP_SECURITY_KEY *key);
```

Description

This service specifies that the previously created key is to be used for encryption and decryption of SNMPv3 response messages.

Note that an authentication key must also be created. SNMPv3 encryption also requires authentication.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
key	Pointer to previously create key.

Return Values

NX_SUCCESS	(0x00)	Successful privacy key set.
NX_NOT_ENABLED	(0x14)	Security not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Use the “my_privacy_key” for the SNMP Agent “my_agent.”   */
status =  nx_snmp_agent_privacy_key_use(&my_agent, &my_privacy_key);


/* If status is NX_SUCCESS the privacy key has been set.  */
```

nx_snmp_agent_sha_key_create

Create SHA key (SNMP v3 only)

Prototype

```
UINT nx_snmp_agent_sha_key_create(NX_SNMP_AGENT *agent_ptr,
                                  UCHAR *password,
                                  NX_SNMP_SECURITY_KEY
                                  *destination_key);
```

Description

This service creates a SHA key that can be used for authentication and encryption.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
password	Pointer to password string.
destination_key	Pointer to SNMP key data structure.

Return Values

NX_SUCCESS	(0x00)	Successful key create.
NX_SNMP_ERROR	(0x100)	Key create error.
NX_PTR_ERROR	(0x07)	Invalid SNMP Agent or key pointer.

Allowed From

Initialization, Threads

Example

```
NX_SNMP_SECURITY_KEY my_key;

/* Create the SHA key for “my_agent.”   */
status =  nx_snmp_agent_sha_key_create(&my_agent, “authpw”, &my_key);


/* If status is NX_SUCCESS the key for the password “authpw” has been created.  */
```

nx_snmp_agent_start

Start SNMP agent

Prototype

```
UINT nx_snmp_agent_start(NX_SNMP_AGENT *agent_ptr);
```

Description

This service binds the UDP socket to the SNMP port 161 and starts the SNMP Agent thread task.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.

Return Values

NX_SUCCESS	(0x00)	Successful start of SNMP Agent.
NX_SNMP_ERROR	(0x100)	SNMP Agent start error.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Start the previously created SNMP Agent “my_agent.”   */
status =  nx_snmp_agent_start(&my_agent);


/* If status is NX_SUCCESS the SNMP Agent “my_agent” has been started.  */
```

nx_snmp_agent_stop

Stop SNMP agent

Prototype

```
UINT nx_snmp_agent_stop(NX_SNMP_AGENT *agent_ptr);
```

Description

This service stops the SNMP Agent thread task and unbinds the UDP socket to the SNMP port.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.

Return Values

NX_SUCCESS	(0x00)	Successful stop of SNMP Agent.
NX_PTR_ERROR	(0x07)	Invalid SNMP Agent pointer.

Allowed From

Threads

Example

```
/* Stop the previously created and started SNMP Agent "my_agent."   */
status =  nx_snmp_agent_stop(&my_agent);

/* If status is NX_SUCCESS the SNMP Agent "my_agent" has been stopped.  */
```

nx_snmp_agent_trap_send

Send SNMPv1 trap

Prototype

```
UINT nx_snmp_agent_trap_send(NX_SNMP_AGENT *agent_ptr,
                             ULONG ip_address, UCHAR *enterprise,
                             UINT trap_type, UINT trap_code,
                             ULONG elapsed_time,
                             NX_SNMP_TRAP_OBJECT *object_list_ptr);
```

Description

This service sends an SNMP trap to the SNMP Manager at the specified IP address. The preferred method for sending an SNMP trap in NetX is to use the *nx_snmp_agent_trap_send* service.
nx_snmp_agent_trap_send is included in NetX to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP address of the SNMP Manager.
enterprise	Enterprise object ID string (sysObectID).
trap_type	Type of trap requested, as follows: NX_SNMP_TRAP_COLDSTART (0) NX_SNMP_TRAP_WARMSTART (1) NX_SNMP_TRAP_LINKDOWN (2) NX_SNMP_TRAP_LINKUP (3) NX_SNMP_TRAP_AUTHENTICATE_FAILURE (4) NX_SNMP_TRAP_EGPNEIGHBORLOSS (5)
trap_code	Specific trap code.
elapsed_time	Time system has been up (sysUpTime).
object_list_ptr	Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL terminated.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP trap send.
NX_SNMP_ERROR	(0x100)	Error sending SNMP trap.
NX_NOT_ENABLED	(0x14)	SNMPv1 not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];

ULONG dest_ip_address = IP_ADDRESS(1,2,3,4);

/* Build list of objects to supply in the trap.  */
trap_list[0].nx_snmp_object_string_ptr =  "1.3.6.1.2.1.2.2.1.1.0";
trap_list[0].nx_snmp_object_data.nx_snmp_object_data_type =  NX_SNMP_INTEGER;
trap_list[0].nx_snmp_object_data.nx_snmp_object_data_msw =   counter;
trap_list[1].nx_snmp_object_string_ptr =  "1.3.6.1.2.1.1.3.0";
trap_list[1].nx_snmp_object_data.nx_snmp_object_data_type =  NX_SNMP_TIME_TICS;
trap_list[1].nx_snmp_object_data.nx_snmp_object_data_msw =   tx_time_get();
trap_list[2].nx_snmp_object_string_ptr =  NX_NULL; /* Terminate list!  */

/* Send trap to SNMP manager at 193.2.2.61.  */
status =  nx_snmp_agent_trap_send(&my_agent,dest_ip_address,
                                 "1.3.6.7.7.7", NX_SNMP_TRAP_LINKUP, counter++,
                                  tx_time_get(), trap_list);

/* If status is NX_SUCCESS the SNMP trap has been sent.  */
```

nx_snmp_agent_trapv2_send

Send SNMPv2 trap

Prototype

```
UINT nx_snmp_agent_trapv2_send(NX_SNMP_AGENT *agent_ptr,
            ULONG ip_address, UCHAR *community, UINT trap_type,
            ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);
```

Description

This service sends an SNMPv2 trap to the SNMP Manager at the specified IP address. The preferred method for sending an SNMP trap in NetX is to use the *nx_snmp_agent_trapv2_send* service. *nx_snmp_agent_trapv2_send* is included in NetX to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP address of the SNMP Manager.
community	Community name (username).
trap_type	Type of trap requested. The standard events are: NX_SNMP_TRAP_COLDSTART (0) NX_SNMP_TRAP_WARMSTART (1) NX_SNMP_TRAP_LINKDOWN (2) NX_SNMP_TRAP_LINKUP (3) NX_SNMP_TRAP_AUTHENTICATE_FAILURE (4) NX_SNMP_TRAP_EGPNEIGHBORLOSS (5) For proprietary data: NX_SNMP_TRAP_CUSTOM (0xFFFFFFFF) (defined in *nx_snmp.h*)
elapsed_time	Time system has been up (sysUpTime).
object_list_ptr	Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL

terminated.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP trap send.
NX_SNMP_ERROR	(0x100)	Error sending SNMP trap.
NX_NOT_ENABLED	(0x14)	SNMPv2 not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];
ULONG  dest_ip_address = IP_ADDRESS(1,2,3,4);

/* Build an empty object list, since it is not needed for the
   NX_SNMP_TRAP_COLDSTART trap.  */
trap_list[0].nx_snmp_object_string_ptr =  NX_NULL;

/* Send trap to SNMP manager at 193.2.2.61.  */
Status =  nx_snmp_agent_trapv2_send(&my_agent, dest_ip_address, "public",
               NX_SNMP_TRAP_COLDSTART, tx_time_get(), trap_list);

/* If status is NX_SUCCESS the SNMP trap has been sent.  */
```

nx_snmp_agent_trapv2_oid_send

Send SNMPv2 trap specifying OID directly

Prototype

```
UINT nx_snmp_agent_trapv2_oid_send(NX_SNMP_AGENT *agent_ptr,
                                   ULONG ip_address, UCHAR *community,
                                   UCHAR *oid, ULONG elapsed_time,
                                   NX_SNMP_TRAP_OBJECT
                                            *object_list_ptr);
```

Description

This service sends an SNMPv2 trap to the SNMP Manager at the specified IP address (IP only) and allows the caller to specify the OID directly. The preferred method for sending an SNMP trap with specified OID in NetX is to use the *nx_snmp_agent_trapv2_oid_send* service. *nx_snmp_agent_trapv2_oid_ send* is included in NetX to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP address of SNMP Manager.
community	Community name (username).
oid	Pointer to buffer containing OID.
elapsed_time	Time system has been up (sysUpTime).
object_list_ptr	Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL terminated.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP trap send.
NX_SNMP_ERROR	(0x100)	Error sending SNMP trap.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or parameter pointer.
NX_IP_ADDRESS_ERROR	(0x21)	Invalid destination IP address.

Allowed From

Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];

/* Build an empty object list */
trap_list[0].nx_snmp_object_string_ptr =  NX_NULL;

/* Send trap to SNMP manager at 193.2.2.61.  */
Status =  nx_snmp_agent_trapv2_oid_send(&my_agent, IP_ADDRESS(193,2,2,61),
                                       "public", (UCHAR *)"0.9.9.9.9.9.9.9.9.9",
                                        tx_time_get(), trap_list);

/* If status is NX_SUCCESS the SNMP trap has been sent.  */
```

nx_snmp_agent_trapv3_send

Send SNMPv3 trap

Prototype

```
UINT nx_snmp_agent_trapv3_send(NX_SNMP_AGENT *agent_ptr,
            ULONG ip_address, UCHAR *community, UINT trap_type,
            ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);
```

Description

This service sends an SNMPv3 trap to the SNMP Manager at the specified address. The preferred method for sending an SNMP trap in NetX is to use the *nx_snmp_agent_trapv3_send* service. *nx_snmp_agent_trapv3_send* is included in NetX to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr Pointer to SNMP Agent control block.

ip_address IP address of the SNMP Manager.

community Community name (username).

trap_type Type of trap requested. The standard events are:

NX_SNMP_TRAP_COLDSTART	(0)
NX_SNMP_TRAP_WARMSTART	(1)
NX_SNMP_TRAP_LINKDOWN	(2)
NX_SNMP_TRAP_LINKUP	(3)
NX_SNMP_TRAP_AUTHENTICATE_FAILURE	(4)
NX_SNMP_TRAP_EGPNEIGHBORLOSS	(5)

For proprietary data:

NX_SNMP_TRAP_CUSTOM (0xFFFFFFFF)
(defined in *nx_snmp.h*)

elapsed_time Time system has been up (sysUpTime).

object_list_ptr Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL terminated.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP trap send.
NX_SNMP_ERROR	(0x100)	Error sending SNMP trap.
NX_NOT_ENABLED	(0x14)	SNMPv3 not enabled.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];
ULONG dest_ip_address = IP_ADDRESS(1,2,3,4);

/* Build an empty object list, since it is not needed for the
   NX_SNMP_TRAP_COLDSTART trap.  */
trap_list[0].nx_snmp_object_string_ptr =  NX_NULL;

/* Send trap to SNMP manager at 193.2.2.61.  */
Status =  nx_snmp_agent_trapv3_send(&my_agent, dest_ip_address, "public",
               NX_SNMP_TRAP_COLDSTART, tx_time_get(), trap_list);

/* If status is NX_SUCCESS the SNMP trap has been sent.  */
```

nx_snmp_agent_trapv3_oid_send

Send SNMPv3 trap specifying OID directly

Prototype

```
UINT nx_snmp_agent_trapv3_oid_send(NX_SNMP_AGENT *agent_ptr,
                                   ULONG ip_address, UCHAR *community,
                                   UCHAR *oid, ULONG elapsed_time,
                                   NX_SNMP_TRAP_OBJECT
                                   *object_list_ptr);
```

Description

This service sends an SNMPv3 trap to the SNMP Manager at the specified IP address (IP only) and allows the caller to specify the OID directly. The preferred method for sending an SNMP trap with specified OID in NetX is to use the *nx_snmp_agent_trapv3_oid_send* service. *nx_snmp_agent_trapv3_oid_ send* is included in NetX to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP address of SNMP Manager.
community	Community name (username).
oid	Pointer to buffer containing OID.
elapsed_time	Time system has been up (sysUpTime).
object_list_ptr	Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL terminated.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP trap send.
NX_SNMP_ERROR	(0x100)	Error sending SNMP trap.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or parameter pointer.
NX_IP_ADDRESS_ERROR	(0x21)	Invalid destination IP address.
NX_OPTION_ERROR	(0x0a)	Invalid parameter.

Allowed From

Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];

/* Build an empty object list */
trap_list[0].nx_snmp_object_string_ptr =  NX_NULL;

/* Send trap to SNMP manager at 193.2.2.61.  */
Status =  nx_snmp_agent_trapv3_oid_send(&my_agent, IP_ADDRESS(193,2,2,61),
                                       "public", (UCHAR *)"0.9.9.9.9.9.9.9.9",
                                        tx_time_get(), trap_list);

/* If status is NX_SUCCESS the SNMP trap has been sent.  */
```

nx_snmp_agent_v3_context_boots_set

Set the number of reboots (if SNMPv3 enabled)

Prototype

```
UINT nx_snmp_agent_v3_context_boots_set(NX_SNMP_AGENT *agent_ptr,
                                        UINT boots);
```

Description

This service sets the number of reboots recorded by the SNMP agent. This service is only available if SNMPv3 is enabled for the SNMP agent because boot count is only used in the SNMPv3 protocol.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block
boots	The value to set SNMP Agent boot count to

Return Values

NX_SUCCESS	(0x00)	Successfully set boot count
NX_SNMP_ERROR	(0x100)	Error setting boot count
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
UINT my_boots = 4;

if (my_agent.nx_snmp_agent_v3_enabled == NX_TRUE)
{
    status = nx_snmp_agent_v3_context_boots_set(&my_agent, my_boots);
}


/* If status is NX_SUCCESS the SNMP boot count is set.  */
```

nx_snmp_object_compare

Compare two objects

Prototype

```
UINT nx_snmp_object_compare(UCHAR *object, UCHAR *reference_object);
```

Description

This service compares the supplied object ID with the reference object ID. Both object IDs are in the ASCII SMI notation, e.g., both object must start with the ASCII string “1.3.6”.

Input Parameters

object	Pointer to object ID.
reference_object	Pointer to the reference object ID.

Return Values

NX_SUCCESS	(0x00)	The object matches the reference object.
NX_SNMP_NEXT_ENTRY	(0x101)	The object is less than the reference object.
NX_SNMP_ERROR	(0x100)	The object is greater than the reference object.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Compare “requested_object” with the sysDescr object ID of
   "1.3.6.1.2.1.1.1.0".  */
Status =  nx_snmp_object_compare(requested_object, "1.3.6.1.2.1.1.1.0");

/* If status is NX_SUCCESS, requested_object is the sysDescr object.
   Otherwise, if status is NX_SNMP_NEXT_ENTRY, the requested object is
   less than the sysDescr. If status is NX_SNMP_ERROR, the object is
   greater than sysDescr. */
```

nx_snmp_object_copy

Copy an object

Prototype

```
UINT nx_snmp_object_copy(UCHAR *source_object_name,
                         UCHAR *destination_object_name);
```

Description

This service copies the source object in ASCII SIM notation to the destination object.

Input Parameters

source_object_name	Pointer to source object ID.
destination_object_name	Pointer to destination object ID.

Return Values

size	Number of bytes copied to destination name. If error, zero is returned.

Allowed From

Initialization, Threads

Example

```
/* Copy "my_object" to "my_new_object".  */
size =  nx_snmp_object_copy(my_object, my_new_object);

/* Size contains the number of bytes copied. */
```

nx_snmp_object_counter_get

Get counter object

Prototype

```
UINT  nx_snmp_object_counter_get(VOID *source_ptr,
                                 NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the counter object at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to counter source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The counter object has be successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Get the ifInOctets (1.3.6.1.2.1.2.2.1.10.0) MIB-2 object.  */
status =  nx_snmp_object_counter_get(&ifInOctets, my_object);

/* If status is NX_SUCCESS, the ifInOctets object has been
   retrieved and is ready to be returned. */
```

nx_snmp_object_counter_set

Set counter object

Prototype

```
UINT  nx_snmp_object_counter_set(VOID *destination_ptr,
                                  NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the counter at the address specified by the destination pointer with the counter value in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to counter destination.
object_data	Pointer to counter source object structure.

Return Values

NX_SUCCESS	(0x00)	The counter object has be successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Set the ifInOctets (1.3.6.1.2.1.2.2.1.10.0) MIB-2 object with
   the counter object value contained in my_object.  */
status =  nx_snmp_object_counter_set(&ifInOctets, my_object);

/* If status is NX_SUCCESS, the ifInOctets object has been
   set. */
```

nx_snmp_object_counter64_get

Get 64-bit counter object

Prototype

```
UINT  nx_snmp_object_counter64_get(VOID *source_ptr,
                                     NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the 64-bit counter object at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to counter source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The counter object has be successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of my_64_bit_counter and place it into my_object
   for return.  */
status =  nx_snmp_object_counter64_get(&my_64_bit_counter, my_object);

/* If status is NX_SUCCESS, the my_64_bit_counter object has been
   retrieved and is ready to be returned. */
```

nx_snmp_object_counter64_set

Set 64-bit counter object

Prototype

```
UINT  nx_snmp_object_counter64_set(VOID *destination_ptr,
                                   NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the 64-bit counter at the address specified by the destination pointer with the counter value in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to counter destination.
object_data	Pointer to counter source object structure.

Return Values

NX_SUCCESS	(0x00)	The counter object has be successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer.

Allowed From

Initialization, Threads

Example

```
/* Set the value of my_64_bit_counter with the value in my_object.  */
status =  nx_snmp_object_counter64_set(&my_64_bit_counter, my_object);

/* If status is NX_SUCCESS, the my_64_bit_counter object has been
   set. */
```

nx_snmp_object_end_of_mib

Set end-of-mib value

Prototype

```
UINT  nx_snmp_object_end_of_mib(VOID *not_used_ptr,
                              NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service creates an object signaling the end of the MIB and is typically called from the GET or GETNEXT application callback routine.

Input Parameters

not_used_ptr	Pointer not used – should be NX_NULL.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The end-of-mib object has be successfully built.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Place an end-of-mib value in my_object.  */
status =  nx_snmp_object_end_of_mib(NX_NULL, my_object);

/* If status is NX_SUCCESS, the my_object is now an end-of-mib object. */
```

nx_snmp_object_gauge_get

Get gauge object

Prototype

```
UINT  nx_snmp_object_gauge_get(VOID *source_ptr,
                               NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the gauge object at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to gauge source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The gauge object has be successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of ifSpeed (1.3.6.1.2.1.2.2.1.5.0) and place it in my_object
   for return.  */
status =  nx_snmp_object_gauge_get(&ifSpeed, my_object);

/* If status is NX_SUCCESS, the my_object now contains the ifSpeed gauge value. */
```

nx_snmp_object_gauge_set

Set gauge object

Prototype

```
UINT  nx_snmp_object_gauge_set(VOID *destination_ptr,
                               NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the gauge at the address specified by the destination pointer with the gauge value in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to gauge destination.
object_data	Pointer to gauge source object structure.

Return Values

NX_SUCCESS	(0x00)	The gauge object has be successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of “my_gauge” from the gauge value in my_object.  */
status =  nx_snmp_object_gauge_set(&my_gauge, my_object);

/* If status is NX_SUCCESS, the my_gauge now contains the new gauge value. */
```

nx_snmp_object_id_get

Get object id

Prototype

```
UINT  nx_snmp_object_id_get(VOID *source_ptr,
                            NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the object ID (in ASCII SIM notation) at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to object ID source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The object ID has be successfully retrieved.
NX_SNMP_ERROR	(0x100)	Invalid length of object
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of sysObjectID(1.3.6.1.2.1.1.2.0) and place it in my_object
   for return.  */
status =  nx_snmp_object_id_get(&sysObjectID, my_object);

/* If status is NX_SUCCESS, the my_object now contains the sysObjectID value. */
```

nx_snmp_object_id_set

Set object id

Prototype

```
UINT  nx_snmp_object_id_set(VOID *destination_ptr,
                            NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the object ID (in ASCII SIM notation) at the address specified by the destination pointer with the object ID in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to object ID destination.
object_data	Pointer to object structure.

Return Values

NX_SUCCESS	(0x00)	The object ID has been successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the string “my_object_id” with the object ID value contained
   in my_object.  */
status =  nx_snmp_object_id_set(my_object_id, my_object);

/* If status is NX_SUCCESS, the my_object_id now contains the object ID value. */
```

nx_snmp_object_integer_get

Get integer object

Prototype

```
UINT  nx_snmp_object_integer_get(VOID *source_ptr,
                                  NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the integer object at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to integer source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The integer object has been successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of sysServices (1.3.6.1.2.1.1.7.0) and place it in my_object
   for return.  */
status =  nx_snmp_object_integer_get(&sysServices, my_object);

/* If status is NX_SUCCESS, the my_object now contains the sysServices value. */
```

nx_snmp_object_integer_set

Set integer object

Prototype

```
UINT  nx_snmp_object_integer_set(VOID *destination_ptr,
                                 NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the integer at the address specified by the destination pointer with the integer value in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to integer destination.
object_data	Pointer to integer source object structure.

Return Values

NX_SUCCESS	(0x00)	The integer object has been successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of ifAdminStatus from the integer value in my_object.  */
status =  nx_snmp_object_integer_set(&ifAdminStatus, my_object);

/* If status is NX_SUCCESS, ifAdnminStatus now contains the new integer value. */
```

nx_snmp_object_ip_address_get

Get IP address object

Prototype

```
UINT  nx_snmp_object_ip_address_get(VOID *source_ptr,
                                     NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the IP address object at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to IP address source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The IP address object has been successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of ipAdEntAddr (1.3.6.1.2.1.4.20.1.1.0) and place it in my_object
   for return.  */
status =  nx_snmp_object_ip_address_get(&ipAdEntAddr, my_object);

/* If status is NX_SUCCESS, the my_object now contains the ipAdEntAddr value. */
```

nx_snmp_object_ip_address_set

Set IP address object

Prototype

```
UINT  nx_snmp_object_ip_address_set(VOID *destination_ptr,
                                     NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the IP address at the address specified by the destination pointer with the IP address in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to IP address to set.
object_data	Pointer to IP address object structure.

Return Values

NX_SUCCESS	(0x00)	The IP address object has been successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of atNetworkAddress to the IP address in my_object.  */
status =  nx_snmp_object_ip_address_set(&atNetworkAddress, my_object);

/* If status is NX_SUCCESS, atNetWorkAddress now contains the new IP address. */
```

nx_snmp_object_no_instance

Set no-instance object

Prototype

```
UINT  nx_snmp_object_no_instance(VOID *not_used_ptr,
                                 NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service creates an object signaling that there was no instance of the specified object and is typically called from the GET or GETNEXT application callback routine.

Input Parameters

not_used_ptr	Pointer not used – should be NX_NULL.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The no-instance object has been successfully built.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Place no-instance value in my_object.  */
status =  nx_snmp_object_no_instance(NX_NULL, my_object);

/* If status is NX_SUCCESS, the my_object is now a no-instance object. */
```

nx_snmp_object_not_found

Set a not-found object

Prototype

```
UINT  nx_snmp_object_not_found(VOID *not_used_ptr,
                               NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service creates an object signaling the object was not found and is typically called from the GET or GETNEXT application callback routine.

Input Parameters

not_used_ptr	Pointer not used – should be NX_NULL.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The not-found object has been successfully built.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Place not-found value in my_object.  */
status =  nx_snmp_object_not_found(NX_NULL, my_object);

/* If status is NX_SUCCESS, the my_object is now a not-found object. */
```

nx_snmp_object_octet_string_get

Get octet string object

Prototype

```
UINT  nx_snmp_object_octet_string_get(VOID *source_ptr,
                                       NX_SNMP_OBJECT_DATA *object_data,
                                       UINT length);
```

Description

This service retrieves the octet string at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to octet string source.
object_data	Pointer to destination object structure.
length	Number of bytes in octet string.

Return Values

NX_SUCCESS	(0x00)	The octet string object has been successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of the 6-byte ifPhysAddress (1.3.6.1.2.1.2.2.1.6.0) and place
   it in my_object for return.  */
status =  nx_snmp_object_octet_string_get(ifPhysAddress, my_object, 6);

/* If status is NX_SUCCESS, the my_object now contains the ifPhysAddress value. */
```

nx_snmp_object_octet_string_set

Set octet string object

Prototype

```
UINT  nx_snmp_object_octet_string_set(VOID *destination_ptr,
                                      NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the octet string at the address specified by the destination pointer with the octet string in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to octet string destination.
object_data	Pointer to octet string source object structure.

Return Values

NX_SUCCESS	(0x00)	The octet string object has been successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of sysContact (1.3.6.1.2.1.1.4.0) from the
   octet string in my_object.  */
status =  nx_snmp_object_octet_string_set(sysContact, my_object);

/* If status is NX_SUCCESS, sysContact now contains the new octet string. */
```

nx_snmp_object_string_get

Get ASCII string object

Prototype

```
UINT  nx_snmp_object_string_get(VOID *source_ptr,
                                NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the ASCII string at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to ASCII string source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The ASCII string object has been successfully retrieved.
NX_SNMP_ERROR	(0x100)	String is too big
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of the sysDescr (1.3.6.1.2.1.1.1.0) and place
   it in my_object for return.  */
status =  nx_snmp_object_string_get(sysDescr, my_object);

/* If status is NX_SUCCESS, the my_object now contains the sysDescr string. */
```

nx_snmp_object_string_set

Set ASCII string object

Prototype

```
UINT  nx_snmp_object_string_set(VOID *destination_ptr,
                                NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the ASCII string at the address specified by the destination pointer with the ASCII string in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to ASCII string destination.
object_data	Pointer to ASCII string source object structure.

Return Values

NX_SUCCESS	(0x00)	The ASCII string object has been successfully set.
NX_SNMP_ERROR	(0x100)	String too large.
NX_SNMP_ERROR_BADVALUE	(0x03)	Invalid character in string
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of sysContact (1.3.6.1.2.1.1.4.0) from the
   ASCII string in my_object.  */
status =  nx_snmp_object_string_set(sysContact, my_object);

/* If status is NX_SUCCESS, sysContact now contains the new ASCII string. */
```

nx_snmp_object_timetics_get

Get timetics object

Prototype

```
UINT  nx_snmp_object_timetics_get(VOID *source_ptr,
                                  NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the timetics at the address specified by the source pointer and places it in the NetX object data structure. This routine is typically called from the GET or GETNEXT application callback routine.

Input Parameters

source_ptr	Pointer to timetics source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The timetics object has been successfully retrieved.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Get the value of the sysUpTime (1.3.6.1.2.1.1.3.0) and place
   it in my_object for return.  */
status =  nx_snmp_object_timetics_get(sysUpTime, my_object);

/* If status is NX_SUCCESS, the my_object now contains the sysUpTime value. */
```

nx_snmp_object_timetics_set

Set timetics object

Prototype

```
UINT  nx_snmp_object_timetics_set(VOID *destination_ptr,
                                  NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the timetics variable at the address specified by the destination pointer with the timetics in the NetX object data structure. This routine is typically called from the SET application callback routine.

Input Parameters

destination_ptr	Pointer to timetics destination.
object_data	Pointer to timetics source object structure.

Return Values

NX_SUCCESS	(0x00)	The timetics object has been successfully set.
NX_SNMP_ERROR_WRONGTYPE	(0x07)	Invalid object type.
NX_PTR_ERROR	(0x07)	Invalid input pointer

Allowed From

Initialization, Threads

Example

```
/* Set the value of “my_time” from the timetics value in my_object.  */
status =  nx_snmp_object_timetics_set(&my_time, my_object);

/* If status is NX_SUCCESS, my_time now contains the new timetics. */
```