Simple Network Management Protocol Agent for NetX Duo (NetX Duo 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 5.0

Contents

Chapter 1

Introduction to NetX Duo SNMP

The Simple Network Management Protocol (SNMP) is a protocol designed for manage devices on the internet. SNMP is a simple protocol that utilizes the connectionless User Datagram Protocol (UDP) services to perform its management function. Because of this, SNMP provides its own reliability mechanism using simple retry logic. SNMP is one of the most used application protocols.

NetX Duo SNMP supports both IPv4 and IPv6 communication with the SNMP Manager. NetX SNMP applications should compile and run in NetX Duo SNMP. However, the developer is encouraged to use the equivalent "duo" service e.g. sending SNMP trap messages. For more details, see "Description of SNMP Agent Services" for nxd_snmp_object_trap_send, *nxd_snmp_object_trapv2_send,* and *nxd_snmp_object_trapv4_send.* Services which are IPv6 only are limited *to nx_snmp_object_ipv6_address_set* and *nx_snmp_object_ipv6_address_get*, also described in "Description of SNMP Agent Services"

The NetX Duo SNMP implementation is that of an SNMP Agent. An agent is responsible for processing a SNMP Manager's commands, which includes responding to them.

NetX Duo SNMP Agent Requirements

In order to function properly, the NetX Duo SNMP package requires that a NetX IP instance has already been created. In addition, UDP must be enabled on that same IP instance.

The NetX Duo SNMP Agent has several additional requirements. First, it requires complete access to UDP *well-known port 161* for handling all SNMP manager requests.

To use NetX Duo SNMP Agent with over IPv6 and to obtain IPv6 objects, IPv6 must be enabled in NetX Duo. Note that NetX Duo SNMP Agent will also support IPv4 communication with the SNMP Manager regardless if IPv6 is enabled in NetX Duo.

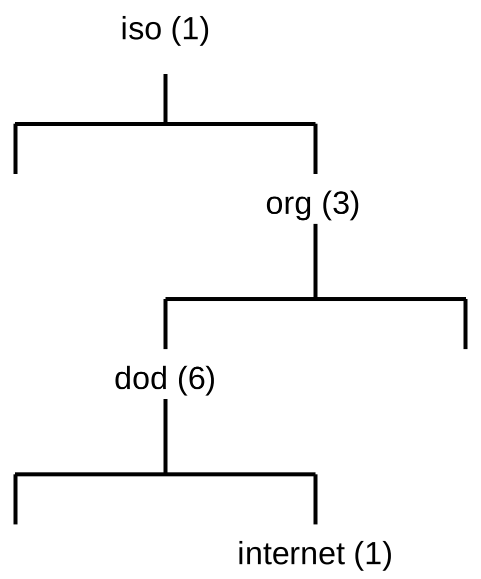
NetX Duo SNMP Constraints

The NetX Duo SNMP protocol implements SNMP version 1, 2, and 3. The SNMP version 3 implementation includes MD5 and SHA authentication, as well as DES encryption. This version of the NetX Duo 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 as 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 Duo 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 the *well-known 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*
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 Duo SNMP Agent processes the request and then calls the corresponding handling routine specified by the application when it called ***nx_snmp_agent_create***.

NetX Duo SNMP Agent Traps

The NetX Duo SNMP Agent provides the ability to also alert an SNMP Manager of a situation 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 at *well-known UDP port 162*.

NetX Duo SNMP Authentication

SNMP authentication is optional and isn’t required in every application. 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 somewhat easy to steal. The SNMP version 3 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 Duo SNMP Agent supports both the standard MD5 and SHA digest algorithms.

When is authentication required? Basically, this is determined by the application in the *snmp_agent_username_process* routine that it specified during the ***nx_snmp_agent_create*** call. It is here the application can reject the supplied username or setup authentication keys and/or privacy keys for the given request. The keys should have been created previously.

NetX Duo SNMP Authentication Callback

As mentioned before, SNMP authentication is optional and isn't required in every application. The NetX Duo SNMP Agent package allows the application to specify (via the ***nx_snmp_agent_create*** call) an authentication callback routine that is called at the beginning of handling each SNMP Client request.

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

For SNMP version 3 situations, the callback routine may also setup authentication and privacy keys during its processing prior to returning a **NX_SUCCESS** for a valid username. Such keys may also be setup outside of the authentication callback if multiple usernames are not supported.

The format of the application authenticate callback routine is very simple and 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.

The return value of the authentication routine specifies whether or not the username is valid. The value **NX_SUCCESS** is returned when the username is valid. Otherwise, **NX_SNMP_ERROR** is returned when the username is invalid.

NetX Duo SNMP Agent GET Callback

The application is responsible for setting up a callback routine responsible for handling GET object requests from the SNMP Manager. The callback is responsible for retrieving the value of the object specified in the request.

The application GET request callback routine is very simple and 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 Duo SNMP API's described below.

If everything is okay, the output of this callback function should be **NX_SUCCESS**. Otherwise, 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 Duo SNMP Agent GETNEXT Callback

The application is responsible for setting up a callback routine responsible for handling GETNEXT object requests from the SNMP Manager. The callback is responsible for retrieving the value of the next object specified by the request.

The application GETNEXT request callback routine is very simple and 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 Duo SNMP API's described below.

If everything is okay, the output of this callback function should be **NX_SUCCESS**. Otherwise, if the callback function can not 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 Duo SNMP Agent SET Callback

The application is responsible for setting up a callback routine responsible for handling SET object requests from the SNMP Manager.

The callback is responsible for setting the value of the object specified by the request.

The application SET request callback routine is very simple and 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 Duo SNMP API's described below.

If everything is okay, the output of this callback function should be **NX_SUCCESS**. Otherwise, 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 Duo SNMP RFCs

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

.

Chapter 2

Installation and Use of the NetX Duo SNMP Agent

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

Product Distribution

SNMP Agent for NetX Duo 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:

nxd_snmp.h	Header file for SNMP for NetX Duo
nxd_snmp.c	C Source file for SNMP Agent for NetX Duo
md5.c	MD5 digest algorithms
sha.c	SHA digest algorithms
des.c	DES encryption algorithms
nxd_snmp.pdf	User Guide for SNMP Agent for NetX Duo
demo_netxduo_snmp.c	Simple SNMP demonstration
demo_netxduo_mib2.c	Simple MIB2 demonstration (MIB has the IPv6 elements)

NetX Duo SNMP Agent Installation

In order to use SNMP for NetX, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory "*\threadx\arm7\green*" then the *nxd_snmp.h*, *nxd_snmp.c*, *md5.c, sha.c* and *des.c* files should be copied into this directory.

Using the NetX Duo SNMP Agent

Using SNMP for NetX Duo is easy. Basically, the application code must include *nxd_snmp.h* after it includes *nx_api.h*. Once *nxd_snmp.h* is included, the application code is then able to make the SNMP function calls specified later in this guide. The application must also include *nxd_snmp.c*, *md5.c, sha.c*, and *des.c* in the build process. These files must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX Duo SNMP.

Note that if **NX_SNMP_NO_SECURITY** is specified in the build process, the *md5.c, sha.c, and des.c* files are not needed.

Note also that since NetX Duo 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 easy it is to use NetX Duo SNMP Agent is described in Figure 1.1 that appears below. In this example, the SNMP include file *nxd_snmp.h* is brought in at line 6. Next, the SNMP Agent is created in "*tx_application_define*" at line 160. Note that the SNMP Agent control block "*my_agent*" was defined as a global variable at line 14 previously. After successful creation, an SNMP Agent is started at line 166. SNMP Manager GET requests are processed starting at line 172, GETNEXT requests are processed starting at line 240, and SET requests are processed starting at line 322. For this example, no authenticate is performed (line 391).

Note that the table shown that implements the SNMP portion of the MIB2 is simply an example. The application could do something completely different, including making string comparisons directly in the GET, GETNEXT, or SET processing.

```
000  /* This is a small demo of the NetX Duo SNMP Agent on the high-performance
001     NetX TCP/IP stack. This demo relies on ThreadX and NetX to show simple
002     SNMP GET/GETNEXT/SET requests on the SNMP MIB-2 objects.  */
003
004  #include  "tx_api.h"
005  #include  "nx_api.h"
006  #include  "nxd_snmp.h"
007
008  #define     DEMO_STACK_SIZE         4096
009
010  /* Define the ThreadX and NetX object control blocks...  */
011
012  NX_PACKET_POOL          pool_0;
013  NX_IP                   ip_0;
014  NX_SNMP_AGENT           my_agent;
015
016
017  /* Define function prototypes.  */
018
019  void    nx_ppc405_driver(NX_IP_DRIVER *driver_req_ptr);
020  UINT    mib2_get_processing(NX_SNMP_AGENT *agent_ptr,
021  UCHAR  *object_requested, NX_SNMP_OBJECT_DATA *object_data);
022  UINT    mib2_getnext_processing(NX_SNMP_AGENT *agent_ptr,
CHAR   *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);


/* Define the SNMP MIB-2 objects.  */

ULONG   snmpInPkts =               0; /* snmpInPkts:Counter                   RO */
ULONG   snmpOutPkts =              0; /* snmpOutPkts:Counter                  RO */
ULONG   snmpInBadVersions =        0; /* snmpInBadVersions:Counter            RO */
ULONG   snmpInBadCommunityNames =  0; /* snmpInBadCommunityNames:Counter      RO */
ULONG   snmpInBadCommunityUsers =  0; /* snmpInBadCommunityUsers:Counter      RO */
ULONG   snmpInASNParseErrs =       0; /* snmpInASNParseErrs:Counter           RO */
ULONG   snmpInTooBigs =            0; /* snmpInTooBigs:Counter                RO */
ULONG   snmpInNoSuchNames =        0; /* snmpInNoSuchNames:Counter            RO */
ULONG   snmpInBadValues =          0; /* snmpInBadValues:Counter              RO */
ULONG   snmpInReadOnlys =          0; /* snmpInReadOnlys:Counter              RO */
ULONG   snmpInGenErrs =            0; /* snmpInGenErrs:Counter                RO */
ULONG   snmpInTotalReqVars =       0; /* snmpInTotalReqVars:Counter           RO */
ULONG   snmpInTotalSetVars =       0; /* snmpInTotalSetVars:Counter           RO */
ULONG   snmpInGetRequests =        0; /* snmpInGetRequests:Counter            RO */
ULONG   snmpInGetNexts =           0; /* snmpInGetNexts:Counter               RO */
ULONG   snmpInSetRequests =        0; /* snmpInSetRequests:Counter            RO */
ULONG   snmpInGetResponses =       0; /* snmpInGetResponses:Counter           RO */
ULONG   snmpInTraps =              0; /* snmpInTraps:Counter                  RO */
ULONG   snmpOutTooBigs =           0; /* snmpOutTooBigs:Counter               RO */
ULONG   snmpOutNoSuchNames =       0; /* snmpOutNoSuchNames:Counter           RO */
ULONG   snmpOutBadValues =         0; /* snmpOutBadValues:Counter             RO */
ULONG   snmpOutGenErrs =           0; /* snmpOutGenErrs:Counter               RO */
ULONG   snmpOutGetRequests =       0; /* snmpOutGetRequests:Counter           RO */
ULONG   snmpOutGetNexts =          0; /* snmpOutGetNexts:Counter              RO */
ULONG   snmpOutSetRequests =       0; /* snmpOutSetRequests:Counter           RO */
ULONG   snmpOutGetResponses =      0; /* snmpOutGetResponses:Counter          RO */
ULONG   snmpOutTraps =             0; /* snmpOutTraps:Counter                 RO */
ULONG   snmpEnableAuthenTraps =    1; /* snmpEnableAuthenTraps:Integer        RW */


/* Define application MIB data structure. Actual application structures
   would certainly vary.  */

typedef struct MIB_ENTRY_STRUCT
{

    UCHAR *object_name;
    void  *object_value_ptr;
    UINT  (*object_get_callback)
          (VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);
    UINT  (*object_set_callback)(VOID *destination_ptr,
NX_SNMP_OBJECT_DATA *object_data);
} MIB_ENTRY;


/* Define the application's MIB-2 objects.  */

MIB_ENTRY   mib2_mib[] = {

    /*    OBJECT ID              OBJECT VARIABLE      GET ROUTINE,SET ROUTINE   */

{"1.3.6.1.2.1.11.1.0", &snmpInPkts,              nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.2.0", &snmpOutPkts,             nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.3.0", &snmpInBadVersions,       nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.4.0", &snmpInBadCommunityNames,nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.5.0", &snmpInBadCommunityUsers,nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.6.0", &snmpInASNParseErrs,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.8.0", &snmpInTooBigs,           nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.9.0", &snmpInNoSuchNames,       nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.10.0",&snmpInBadValues,         nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.11.0",&snmpInReadOnlys,         nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.12.0",&snmpInGenErrs,           nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.13.0",&snmpInTotalReqVars,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.14.0",&snmpInTotalSetVars,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.15.0",&snmpInGetRequests,       nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.16.0",&snmpInGetNexts,          nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.17.0",&snmpInSetRequests,       nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.18.0",&snmpInGetResponses,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.19.0",&snmpInTraps,             nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.20.0",&snmpOutTooBigs,          nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.21.0",&snmpOutNoSuchNames,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.22.0",&snmpOutBadValues,        nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.24.0",&snmpOutGenErrs,          nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.25.0",&snmpOutGetRequests,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.26.0",&snmpOutGetNexts,         nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.27.0",&snmpOutSetRequests,      nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.28.0",&snmpOutGetResponses,     nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.29.0",&snmpOutTraps,            nx_snmp_object_counter_get,NX_NULL},
{"1.3.6.1.2.1.11.30.0",&snmpEnableAuthenTraps,  nx_snmp_object_integer_get,
    nx_snmp_object_integer_set},
{"1.3.6.1.6",          "1.3.6.1.6",              nx_snmp_object_end_of_mib, NX_NULL},
{NX_NULL, NX_NULL, NX_NULL, NX_NULL}
};


/* 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;

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

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

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

    /* Create an IP instance.  */
    nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(192, 2, 2, 187),
                        0xFFFFFF00UL, &pool_0, nx_ppc405_driver,
                        pointer, 4096, 1);
    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.  */
    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);
    pointer =  pointer + 4096;

    /* 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 1.1 Example of SNMP Agent use with NetX Duo

Configuration Options

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

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic SNMP error checking. It is typically used after the application has been debugged.
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 *nxd_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 *nxd_snmp.h*.
NX_SNMP_TIME_TO_LIVE	Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80, but can be redefined prior to inclusion of *nxd_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

	nxd_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 *nxd_snmp.h.*
NX_SNMP_MAX_CONTEXT_STRING	Specifies the number of bytes allowed for a context engine string in the SNMP Agent. The default value is set to 32, but can be redefined prior to inclusion of *nxd_snmp.h.*
NX_SNMP_MAX_USER_NAME	Specifies the number of bytes allowed in a username (including community strings). The default value is set to 64, but can be redefined prior to inclusion of *nxd_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 *nxd_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 *nxd_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 *nxd_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 *nxd_snmp.h.*
NX_SNMP_DISABLE_V1	Defined, this removes all the SNMP Version 1 processing in *nxd_snmp.c*.
NX_SNMP_DISABLE_V2	Defined, this removes all the SNMP Version 2 processing in *nxd_snmp.c*.
NX_SNMP_DISABLE_V3	Defined, this removes all the SNMP Version 3 processing in *nxd_snmp.c*.

Chapter 3

Description of SNMP Agent Services

This chapter contains a description of all NetX Duo 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_start
Start SNMP agent

nx_snmp_agent_stop
Stop SNMP agent

nx_snmp_agent_trap_send
Send SNMP v1 trap (IPv4 only)

nx_snmp_agent_trapv2_send
Send SNMP v2 trap (IPv4 only)

nx_snmp_agent_trapv3_send
Send SNMP v3 trap (IPv4 only)

nxd_snmp_agent_trap_send
Send SNMP v1 trap (IPv4 and IPv6)

nxd_snmp_agent_trapv2_send
Send SNMP v2 trap (IPv4 and IPv6)

nxd_snmp_agent_trapv3_send
Send SNMP v3 trap (IPv4 and IPv6)

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 (IPv4 only)

nx_snmp_object_ip_address_set
Set IP address object (IPv4 only)

nx_snmp_object_ipv6_address_get
Get IP address object (IPv6 only)

nx_snmp_object_ipv6_address_set
Set IP address object (IPv6 only)

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_authenticate_key_use

Specify authentication key (SNMP v3)

Prototype

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

Description

This service specifies the key to be used for authentication for all requests made after it is set. Supplying a NX_NULL value for the key disables authentication.

Note: The key must be created with one of the key creation services prior to calling this routine.

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 setup.
NX_PTR_ERROR	(0x16)	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 authentication of requests.  */
```

See Also

nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_agent_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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 that was last supplied to the SNMP Agent (which is effectively the username for SNMP versions 1 and 2).

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
community_string_ptr	Pointer to an application string pointer. The actual string resides in the SNMP Agent control block.

Return Values

NX_SUCCESS	(0x00)	Successful SNMP community get.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or community string 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use,

nx_snmp_agent_context_engine_set,nx_snmp_agent_context_name_set,
nx_snmp_agent_create, nx_snmp_agent_delete,
nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use,
nx_snmp_agent_sha_key_create, nx_snmp_agent_start,
nx_snmp_agent_stop, snmp_agent_trap_send,
nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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 SNMP version 3 processing.

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_SNMP_ERROR	(0x100)	Context engine size error.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or context engine pointer.

Allowed From

Initialization, Threads

Example

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

/* 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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 SNMP version 3 processing.

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	(0x16)	Invalid SNMP Agent or context name 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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	(0x16)	Invalid SNMP Agent, IP instance, stack, or function 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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_SNMP_ERROR	(0x100)	SNMP Agent delete error.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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 Duo 5.6 or higher which support multihoming. 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_NOT_ENABLED	(0x14)	SNMP host interface not enabled for multihome support.
NX_SNMP_ERROR	(0x100)	SNMP Agent delete error.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent pointer.
NX_INVALID_INTERFACE	(0x4C)	Invalid index specified

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 been deleted.  */
```

See Also

nx_snmp_agent_create, nx_snmp_agent_delete

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_SNMP_ERROR	(0x100)	Key create error.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or key 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set,

nx_snmp_agent_create, nx_snmp_agent_delete,
nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create,
nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send,
nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nx_snmp_agent_privacy_key_use

Specify encryption key (SNMP v3 only)

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.

Note that a authentication key must have previously been specified. SNMP v3 does not allow privacy (encryption) without 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 setup.
NX_SNMP_ERROR	(0x100)	Error setting up privacy key.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent or key 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get,

nx_snmp_agent_context_engine_set, nx_snmp_context_name_set,
nx_snmp_agent_create, nx_snmp_agent_delete,
nx_snmp_agent_md5_key_create, nx_snmp_agent_sha_key_create,
nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send,
nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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 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_SNMP_ERROR	(0x100)	Key create error.
NX_PTR_ERROR	(0x16)	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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set,

nx_snmp_agent_create, nx_snmp_agent_delete,
nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use,
nx_snmp_agent_start, nx_snmp_agent_stop, snmp_agent_trap_send,
nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nx_snmp_agent_start

Start SNMP agent

Prototype

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

Description

This service starts the SNMP Agent.

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	(0x16)	Invalid SNMP Agent 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_stop, nx_snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.

Return Values

NX_SUCCESS	(0x00)	Successful stop of SNMP Agent.
NX_SNMP_ERROR	(0x100)	SNMP Agent stop error.
NX_PTR_ERROR	(0x16)	Invalid SNMP Agent pointer.

Allowed From

Initialization, 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nx_snmp_agent_trap_send

Send SNMP v1 trap *(IPv4 only)*

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 IPv4 address. The preferred method for sending an SNMP trap in NetX Duo is to use the *nxd_snmp_agent_trap_send* service. *nx_snmp_agent_trap_send* is included in NetX Duo to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IPv4 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_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

Initialization, 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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nxd_snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nxd_snmp_agent_trap_send

Send SNMP v1 trap *(IPv4 and IPv6)*

Prototype

```
UINT nxd_snmp_agent_trap_send(NX_SNMP_AGENT *agent_ptr,
            NXD_ADDRESS &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 equivalent method for sending an SNMP trap in NetX is the *nxd_snmp_agent_trap_send* service. .

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IPv4 or IPv6 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_INVALID_IP_PROTOCOL_ERROR		

	(0x104)	Unsupported IP version
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

Initialization, Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];

NXD_ADDRESS dest_ip_address;

    dest_ip_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    dest_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
    dest_ip_address.nxd_ip_address.v6[1] = 0xf101;
    dest_ip_address.nxd_ip_address.v6[2] = 0x00000000;
    dest_ip_address.nxd_ip_address.v6[3] = 0x00000101;

/* 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 =  nxd_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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nx_snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nx_snmp_agent_trapv2_send

Send SNMP v2 trap (IPv4 only)

Prototype

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

Description

This service sends an SNMP V2 trap to the SNMP Manager at the specified IPv4 address. The preferred method for sending an SNMP trap in NetX Duo is to use the *nxd_snmp_agent_trapv2_send* service. *nx_snmp_agent_trapv2_send* is included in NetX Duo to support existing NetX SNMP Agent applications.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IPv4 address of the SNMP Manager.
community	Community name (username).
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)
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

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_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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nx_snmp_agent_trap_send, nxd_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nxd_snmp_agent_trapv2_send

Send SNMP v2 trap (IPv4 and IPv6)

Prototype

```
UINT nxd_snmp_agent_trapv2_send(NX_SNMP_AGENT *agent_ptr,
            NXD_ADDRESS *ip_address, UCHAR *community, UINT trap_type,
            ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);
```

Description

This service sends an SNMP V2 trap to the SNMP Manager at the specified IP address. .

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP (IPv4 or IPv6) address of the SNMP Manager.
community	Community name (username).
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)
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_SNMP_INVALID_IP_PROTOCOL_ERROR	(0x104)	Unsupported IP version
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

Initialization, Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];
NXD_ADDRESS dest_ip_address;

    dest_ip_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    dest_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
    dest_ip_address.nxd_ip_address.v6[1] = 0xf101;
    dest_ip_address.nxd_ip_address.v6[2] = 0x00000000;
    dest_ip_address.nxd_ip_address.v6[3] = 0x00000101;

/* 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 =  nxd_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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nx_snmp_agent_trap_send, nx_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

nx_snmp_agent_trapv3_send

Send SNMP v3 trap (IPv4 only)

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 SNMP trap to the SNMP Manager at the specified IPv4 address. The preferred method for sending an SNMP trap in NetX Duo is to use the *nxd_snmp_agent_trapv3_send* service. *nx_snmp_agent_trapv3_send* is included in NetX Duo to support existing NetX SNMP Agent applications.

This trap is basically the same as the SNMP v2 trap, except the trap message format is contained in the SNMP v3 PDU.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IPv4 address of the SNMP Manager.
community	Community name (username).
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)
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

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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nxd_snmp_agent_trap_send, nxd_snmp_agent_trapv2_send, nxd_snmp_agent_trapv3_send

nxd_snmp_agent_trapv3_send

Send SNMP v3 trap (IPv4 and IPv6)

Prototype

```
UINT nxd_snmp_agent_trapv3_send(NX_SNMP_AGENT *agent_ptr,
            NXD_ADDRESS *ip_address, UCHAR *community, UINT trap_type,
            ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);
```

Description

This service sends an SNMP trap to the SNMP Manager at the specified IPv4 address. This trap is basically the same as the SNMP v2 trap, except the trap message format is contained in the SNMP v3 PDU.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP (IPv4 or IPv6) address of the SNMP Manager.
community	Community name (username).
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)
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_SNMP_INVALID_IP_PROTOCOL_ERROR	(0x104)	Unsupported IP version
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

Initialization, Threads

Example

```
NX_SNMP_TRAP_OBJECT trap_list[5];
NXD_ADDRESS dest_ip_address;

    dest_ip_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    dest_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
    dest_ip_address.nxd_ip_address.v6[1] = 0xf101;
    dest_ip_address.nxd_ip_address.v6[2] = 0x00000000;
    dest_ip_address.nxd_ip_address.v6[3] = 0x00000101;

/* 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 =  nxd_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.  */
```

See Also

nx_snmp_agent_authenticate_key_use, nx_snmp_agent_community_get, nx_snmp_agent_context_engine_set, nx_snmp_context_name_set, nx_snmp_agent_create, nx_snmp_agent_delete, nx_snmp_agent_md5_key_create, nx_snmp_agent_privacy_key_use, nx_snmp_agent_sha_key_create, nx_snmp_agent_start, nx_snmp_agent_stop, nxd_snmp_agent_trap_send, nxd_snmp_agent_trapv2_send, nx_snmp_agent_trapv3_send

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	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_copy, nx_snmp_object_counter_get,
nx_snmp_object_counter_set, nx_snmp_object_counter64_get,
nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib,
nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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.

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get, nx_snmp_object_id_set, nx_snmp_object_integer_get, nx_snmp_object_integer_set, nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set, nx_snmp_object_no_instance, nx_snmp_object_not_found, nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set, nx_snmp_object_string_get, nx_snmp_object_string_set, nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_set, nx_snmp_object_counter64_get,

nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib,
nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter64_get,
nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib,
nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_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	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib,
nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_get, nx_snmp_object_end_of_mib,
nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_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	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set,

nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_set,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_id_get, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set,

nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,
nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_set,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,
nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_integer_get, nx_snmp_object_integer_set,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set,

nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get, nx_snmp_object_id_set, nx_snmp_object_integer_set, nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set, nx_snmp_object_no_instance, nx_snmp_object_not_found, nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set, nx_snmp_object_string_get, nx_snmp_object_string_set, nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set,
nx_snmp_object_no_instance, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

nx_snmp_object_ip_address_get

Get IP address object (IPv4 only)

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 IPv4 address source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The IP address object has be successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,

nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get, nx_snmp_object_id_set, nx_snmp_object_integer_get, nx_snmp_object_integer_set, nx_snmp_object_ip_address_set, nx_snmp_object_ipv6_address_set, nx_snmp_object_no_instance, nx_snmp_object_not_found, nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set, nx_snmp_object_string_get, nx_snmp_object_string_set, nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

nx_snmp_object_ipv6_address_get

Get IP address object (IPv6 only)

Prototype

```
UINT  nx_snmp_object_ipv6_address_get(VOID *source_ptr,
                                      NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service retrieves the IPv6 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 IPv6 address source.
object_data	Pointer to destination object structure.

Return Values

NX_SUCCESS	(0x00)	The IP address object has be successfully retrieved.
NX_SNMP_ERROR_WRONGTYPE	**(0x07)**	Incorrect input SNMP object Code
NX_NOT_ENABLED	(0x14)	IPv6 not enabled
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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_ipv6_address_get(&ipAdEntAddr, my_object);

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

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,
nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_get,
nx_snmp_object_string_set, nx_snmp_object_timetics_get,
nx_snmp_object_timetics_set

nx_snmp_object_ip_address_set

Set IPv4 address object

Prototype

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

Description

This service sets the IPv4 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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ipv6_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_get,
nx_snmp_object_string_set, nx_snmp_object_timetics_get,
nx_snmp_object_timetics_set

nx_snmp_object_ipv6_address_set

Set IPv6 address object

Prototype

```
UINT  nx_snmp_object_ipv6_address_set(VOID *destination_ptr,
                                      NX_SNMP_OBJECT_DATA *object_data);
```

Description

This service sets the IPv6 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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

Allowed From

Initialization, Threads

Example

```
/* Set the value of atNetworkAddress to the IP address in my_object.  */
status =  nx_snmp_object_ipv6_address_set(&atNetworkAddress, my_object);

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

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get, nx_snmp_object_id_set, nx_snmp_object_integer_get, nx_snmp_object_integer_set, nx_snmp_object_ip_address_set, nx_snmp_object_ipv6_address_get, nx_snmp_object_no_instance, nx_snmp_object_not_found, nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set, nx_snmp_object_string_get, nx_snmp_object_string_set, nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 was be successfully built.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,

nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_not_found,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

nx_snmp_object_not_found

Set 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 was be successfully built.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get,

nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_octet_string_get, nx_snmp_object_octet_string_set,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set,

nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get, nx_snmp_object_gauge_set, nx_snmp_object_id_get, nx_snmp_object_id_set, nx_snmp_object_integer_get, nx_snmp_object_integer_set, nx_snmp_object_ip_address_get, nx_snmp_object_ip_address_set, nx_snmp_object_no_instance, nx_snmp_object_not_found, nx_snmp_object_octet_string_set, nx_snmp_object_string_get, nx_snmp_object_string_set, nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_string_get, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy,
nx_snmp_object_counter_get, nx_snmp_object_counter_set,
nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,
nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,

nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_set,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_get,
nx_snmp_object_timetics_get, nx_snmp_object_timetics_set

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 be successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set, nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,

nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_get,
nx_snmp_object_string_set, nx_snmp_object_timetics_set

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 be successfully set.
NX_SNMP_ERROR	(0x100)	Invalid object type.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

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. */
```

See Also

nx_snmp_object_compare, nx_snmp_object_copy, nx_snmp_object_counter_get, nx_snmp_object_counter_set, nx_snmp_object_counter64_get, nx_snmp_object_counter64_set,

nx_snmp_object_end_of_mib, nx_snmp_object_gauge_get,
nx_snmp_object_gauge_set, nx_snmp_object_id_get,
nx_snmp_object_id_set, nx_snmp_object_integer_get,
nx_snmp_object_integer_set, nx_snmp_object_ip_address_get,
nx_snmp_object_ip_address_set, nx_snmp_object_no_instance,
nx_snmp_object_not_found, nx_snmp_object_octet_string_get,
nx_snmp_object_octet_string_set, nx_snmp_object_string_get,
nx_snmp_object_string_set, nx_snmp_object_timetics_get