



the high-performance real-time implementation
of TCP/IP standards

Simple Network Management Protocol (SNMP)

Agent

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

©2002-2011 by Express Logic, Inc.

All rights reserved. This document and the associated NetX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden. Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

Trademarks

NetX, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

Warranty Limitations

Express Logic, Inc. makes no warranty of any kind that the NetX products will meet the USER's requirements, or will operate in the manner specified by the USER, or that the operation of the NetX products will operate uninterrupted or error free, or that any defects that may exist in the NetX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the NetX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1054

Revision 5.4

Contents

Chapter 1 Introduction to SNMP	5
SNMP Agent Requirements	5
SNMP Constraints.....	5
SNMP Object Names	6
SNMP Manager Requests.....	6
SNMP Agent Traps	7
SNMP Authentication	7
SNMP Authentication Callback.....	8
SNMP Agent GET Callback	9
SNMP Agent GETNEXT Callback.....	9
SNMP Agent SET Callback	10
SNMP RFCs.....	11
Chapter 2 Installation and Use of SNMP Agent	12
Product Distribution	12
SNMP Agent Installation	12
Using SNMP Agent	12
Small Example System	13
Configuration Options.....	20
Chapter 3 Description of SNMP Agent Services	23
nx_snmp_agent_authenticate_key_use	26
nx_snmp_agent_community_get.....	28
nx_snmp_agent_context_engine_set.....	30
nx_snmp_agent_context_name_set.....	32
nx_snmp_agent_create	34
nx_snmp_agent_delete	36
nx_snmp_agent_md5_key_create	37
nx_snmp_agent_privacy_key_use	39
nx_snmp_agent_sha_key_create.....	41
nx_snmp_agent_start.....	43
nx_snmp_agent_stop	44
nx_snmp_agent_trap_send.....	45
nx_snmp_agent_trapv2_send	47
nx_snmp_agent_trapv2_oid_send	49
nx_snmp_agent_trapv3_send	51
nx_snmp_agent_trapv3_oid_send	53
nx_snmp_object_compare	55
nx_snmp_object_copy.....	57
nx_snmp_object_counter_get	59
nx_snmp_object_counter_set	61
nx_snmp_object_counter64_get	63
nx_snmp_object_counter64_set.....	65

nx_snmp_object_end_of_mib.....	67
nx_snmp_object_gauge_get	69
nx_snmp_object_gauge_set	71
nx_snmp_object_id_get	73
nx_snmp_object_id_set.....	75
nx_snmp_object_integer_get	77
nx_snmp_object_integer_set	79
nx_snmp_object_ip_address_get.....	81
nx_snmp_object_ip_address_set.....	83
nx_snmp_object_no_instance.....	85
nx_snmp_object_not_found	87
nx_snmp_object_octet_string_get.....	89
nx_snmp_object_octet_string_set.....	91
nx_snmp_object_string_get	93
nx_snmp_object_string_set.....	95
nx_snmp_object_timetics_get	97
nx_snmp_object_timetics_set	99

Chapter 1

Introduction to 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.

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

SNMP Agent Requirements

In order to function properly, the NetX 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 SNMP Agent has several additional requirements. First, it requires complete access to UDP *well-known port 161* for handling all SNMP manager requests.

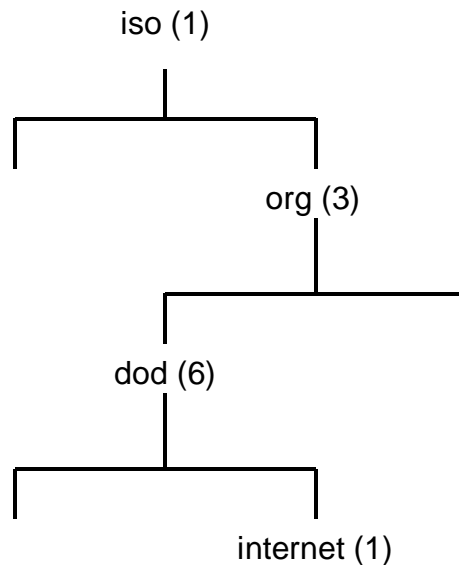
SNMP Constraints

The NetX 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 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 is a set of objects that are defined by the Structure of Management Information (SMI) as defined by RFC 1155. The structure is a hierarchical tree type of structure that looks like the following:



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

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

SNMP Manager Requests

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

SNMP Command	Meaning
GET	<i>Get the specified object</i>
GETNEXT	<i>Get the next logical object after the specified object ID</i>
SET	<i>Set the specified object</i>

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

SNMP Agent Traps

The NetX 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*.

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 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.

SNMP Authentication Callback

As mentioned before, SNMP authentication is optional and isn't required in every application. The NetX 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 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
<i>agent_ptr</i>	Pointer to calling SNMP agent.
<i>username</i>	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.

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

If 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.

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

If 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.

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
-----------	---------

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

If 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.

SNMP RFCs

NetX 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 SNMP Agent

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

Product Distribution

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

<code>nx_snmp.h</code>	Header file for SNMP for NetX
<code>nx_snmp.c</code>	C Source file for SNMP Agent for NetX
<code>md5.c</code>	MD5 digest algorithms
<code>sha.c</code>	SHA digest algorithms
<code>des.c</code>	DES encryption algorithms
<code>nx_snmp.pdf</code>	PDF description of SNMP Agent for NetX
<code>demo_netx_snmp.c</code>	Simple SNMP demonstration
<code>demo_netx_mib2.c</code>	Simple MIB2 demonstration

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 `nx_snmp.h`, `nx_snmp.c`, `md5.c`, `sha.c` and `des.c` files should be copied into this directory.

Using SNMP Agent

Using SNMP for NetX is easy. Basically, the application code must include `nx_snmp.h` after it includes `nx_api.h`. Once `nx_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 `nx_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 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 SNMP utilizes NetX 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 SNMP Agent is described in Figure 1.1 that appears below. In this example, the SNMP include file *nx_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 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 "nx_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,
023    CHAR *object_requested, NX_SNMP_OBJECT_DATA *object_data);
024 UINT    mib2_set_processing(NX_SNMP_AGENT *agent_ptr,
025    UCHAR *object_requested, NX_SNMP_OBJECT_DATA *object_data);

```

```

026 UINT    mib2_username_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *username);
027 VOID    mib2_variable_update(NX_IP *ip_ptr, NX_SNMP_AGENT *agent_ptr);
028
029
030 /* Define the SNMP MIB-2 objects. */
031
032 ULONG    snmpInPkts =          0; /* snmpInPkts:Counter          RO */
033 ULONG    snmpOutPkts =         0; /* snmpOutPkts:Counter         RO */
034 ULONG    snmpInBadVersions =   0; /* snmpInBadVersions:Counter   RO */
035 ULONG    snmpInBadCommunityNames = 0; /* snmpInBadCommunityNames:Counter RO */
036 ULONG    snmpInBadCommunityUsers = 0; /* snmpInBadCommunityUsers:Counter RO */
037 ULONG    snmpInASNParseErrs =  0; /* snmpInASNParseErrs:Counter  RO */
038 ULONG    snmpInTooBigs =       0; /* snmpInTooBigs:Counter       RO */
039 ULONG    snmpInNoSuchNames =   0; /* snmpInNoSuchNames:Counter   RO */
040 ULONG    snmpInBadValues =     0; /* snmpInBadValues:Counter     RO */
041 ULONG    snmpInReadOnlys =     0; /* snmpInReadOnlys:Counter     RO */
042 ULONG    snmpInGenErrs =       0; /* snmpInGenErrs:Counter       RO */
043 ULONG    snmpInTotalReqVars =  0; /* snmpInTotalReqVars:Counter  RO */
044 ULONG    snmpInTotalSetVars =  0; /* snmpInTotalSetVars:Counter  RO */
045 ULONG    snmpInGetRequests =   0; /* snmpInGetRequests:Counter   RO */
046 ULONG    snmpInGetNexts =      0; /* snmpInGetNexts:Counter      RO */
047 ULONG    snmpInSetRequests =   0; /* snmpInSetRequests:Counter   RO */
048 ULONG    snmpInGetResponses =  0; /* snmpInGetResponses:Counter  RO */
049 ULONG    snmpInTraps =         0; /* snmpInTraps:Counter         RO */
050 ULONG    snmpOutTooBigs =       0; /* snmpOutTooBigs:Counter      RO */
051 ULONG    snmpOutNoSuchNames =  0; /* snmpOutNoSuchNames:Counter  RO */
052 ULONG    snmpOutBadValues =    0; /* snmpOutBadValues:Counter    RO */
053 ULONG    snmpOutGenErrs =      0; /* snmpOutGenErrs:Counter      RO */
054 ULONG    snmpOutGetRequests =  0; /* snmpOutGetRequests:Counter  RO */
055 ULONG    snmpOutGetNexts =     0; /* snmpOutGetNexts:Counter     RO */
056 ULONG    snmpOutSetRequests =  0; /* snmpOutSetRequests:Counter  RO */
057 ULONG    snmpOutGetResponses =  0; /* snmpOutGetResponses:Counter RO */
058 ULONG    snmpOutTraps =        0; /* snmpOutTraps:Counter        RO */
059 ULONG    snmpEnableAuthenTraps = 1; /* snmpEnableAuthenTraps:Integer RW */
060
061
062 /* Define application MIB data structure. Actual application structures
063    would certainly vary. */
064
065 typedef struct MIB_ENTRY_STRUCT
066 {
067     UCHAR *object_name;
068     void *object_value_ptr;
069     UINT (*object_get_callback)
070         (VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);
071     UINT (*object_set_callback)(VOID *destination_ptr,
072     NX_SNMP_OBJECT_DATA *object_data);
073 } MIB_ENTRY;
074
075
076 /* Define the application's MIB-2 objects. */
077
078 MIB_ENTRY mib2_mib[] = {
079
080     /* OBJECT ID OBJECT VARIABLE GET ROUTINE,SET ROUTINE */
081
082     {"1.3.6.1.2.1.11.1.0", &snmpInPkts, nx_snmp_object_counter_get,NX_NULL},
083     {"1.3.6.1.2.1.11.2.0", &snmpOutPkts, nx_snmp_object_counter_get,NX_NULL},
084     {"1.3.6.1.2.1.11.3.0", &snmpInBadVersions, nx_snmp_object_counter_get,NX_NULL},
085     {"1.3.6.1.2.1.11.4.0", &snmpInBadCommunityNames,nx_snmp_object_counter_get,NX_NULL},
086     {"1.3.6.1.2.1.11.5.0", &snmpInBadCommunityUsers,nx_snmp_object_counter_get,NX_NULL},
087     {"1.3.6.1.2.1.11.6.0", &snmpInASNParseErrs, nx_snmp_object_counter_get,NX_NULL},
088     {"1.3.6.1.2.1.11.8.0", &snmpInTooBigs, nx_snmp_object_counter_get,NX_NULL},
089     {"1.3.6.1.2.1.11.9.0", &snmpInNoSuchNames, nx_snmp_object_counter_get,NX_NULL},
090     {"1.3.6.1.2.1.11.10.0",&snmpInBadValues, nx_snmp_object_counter_get,NX_NULL},
091     {"1.3.6.1.2.1.11.11.0",&snmpInReadOnlys, nx_snmp_object_counter_get,NX_NULL},
092     {"1.3.6.1.2.1.11.12.0",&snmpInGenErrs, nx_snmp_object_counter_get,NX_NULL},
093     {"1.3.6.1.2.1.11.13.0",&snmpInTotalReqVars, nx_snmp_object_counter_get,NX_NULL},
094     {"1.3.6.1.2.1.11.14.0",&snmpInTotalSetVars, nx_snmp_object_counter_get,NX_NULL},
095     {"1.3.6.1.2.1.11.15.0",&snmpInGetRequests, nx_snmp_object_counter_get,NX_NULL},
096     {"1.3.6.1.2.1.11.16.0",&snmpInGetNexts, nx_snmp_object_counter_get,NX_NULL},

```

```

097 {"1.3.6.1.2.1.11.17.0",&snmpInSetRequests, nx_snmp_object_counter_get,NX_NULL},
098 {"1.3.6.1.2.1.11.18.0",&snmpInGetResponses, nx_snmp_object_counter_get,NX_NULL},
099 {"1.3.6.1.2.1.11.19.0",&snmpInTraps, nx_snmp_object_counter_get,NX_NULL},
100 {"1.3.6.1.2.1.11.20.0",&snmpOutTooBigs, nx_snmp_object_counter_get,NX_NULL},
101 {"1.3.6.1.2.1.11.21.0",&snmpOutNoSuchNames, nx_snmp_object_counter_get,NX_NULL},
102 {"1.3.6.1.2.1.11.22.0",&snmpOutBadValues, nx_snmp_object_counter_get,NX_NULL},
103 {"1.3.6.1.2.1.11.24.0",&snmpOutGenErrs, nx_snmp_object_counter_get,NX_NULL},
104 {"1.3.6.1.2.1.11.25.0",&snmpOutGetRequests, nx_snmp_object_counter_get,NX_NULL},
105 {"1.3.6.1.2.1.11.26.0",&snmpOutGetNexts, nx_snmp_object_counter_get,NX_NULL},
106 {"1.3.6.1.2.1.11.27.0",&snmpOutSetRequests, nx_snmp_object_counter_get,NX_NULL},
107 {"1.3.6.1.2.1.11.28.0",&snmpOutGetResponses, nx_snmp_object_counter_get,NX_NULL},
108 {"1.3.6.1.2.1.11.29.0",&snmpOutTraps, nx_snmp_object_counter_get,NX_NULL},
109 {"1.3.6.1.2.1.11.30.0",&snmpEnableAuthenTraps, nx_snmp_object_integer_get,
110 nx_snmp_object_integer_set},
111 {"1.3.6.1.6", "1.3.6.1.6", nx_snmp_object_end_of_mib, NX_NULL},
112 {NX_NULL, NX_NULL, NX_NULL, NX_NULL}
113 };
114
115
116 /* Define main entry point. */
117
118 int main()
119 {
120
121     /* Enter the ThreadX kernel. */
122     tx_kernel_enter();
123 }
124
125
126 /* Define what the initial system looks like. */
127 void tx_application_define(void *first_unused_memory)
128 {
129
130     UCHAR *pointer;
131
132     /* Setup the working pointer. */
133     pointer = (UCHAR *) first_unused_memory;
134
135     /* Initialize the NetX system. */
136     nx_system_initialize();
137
138     /* Create packet pool. */
139     nx_packet_pool_create(&pool_0, "NetX Packet Pool 0",
140                          2048, pointer, 20000);
141     pointer = pointer + 20000;
142
143     /* Create an IP instance. */
144     nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(192, 2, 2, 187),
145                0xFFFFFFFFUL, &pool_0, nx_ppc405_driver,
146                pointer, 4096, 1);
147     pointer = pointer + 4096;
148
149     /* Enable ARP and supply ARP cache memory for IP Instance 0. */
150     nx_arp_enable(&ip_0, (void *) pointer, 1024);
151     pointer = pointer + 1024;
152
153     /* Enable UDP processing for IP instance. */
154     nx_udp_enable(&ip_0);
155
156     /* Enable ICMP for ping. */
157     nx_icmp_enable(&ip_0);
158
159     /* Create an SNMP agent instance. */
160     nx_snmp_agent_create(&my_agent, "SNMP Agent", &ip_0, pointer, 4096, &pool_0,
161                        mib2_username_processing, mib2_get_processing, mib2_getnext_processing,
162                        mib2_set_processing);
163     pointer = pointer + 4096;
164
165     /* Start the SNMP instance. */
166     nx_snmp_agent_start(&my_agent);
167 }
168

```

```

169
170 /* Define the application's GET processing routine. */
171
172 UINT mib2_get_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
173 NX_SNMP_OBJECT_DATA *object_data)
174 {
175
176     UINT    i;
177     UINT    status;
178
179
180     printf("SNMP Manager GET Request For: %s", object_requested);
181
182     /* Loop through the sample MIB to see if we have information for the
183 supplied variable. */
184     i = 0;
185     status = NX_SNMP_ERROR;
186     while (mib2_mib[i].object_name)
187     {
188
189         /* See if we have found the matching entry. */
190         status = nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);
191
192         /* Was it found? */
193         if (status == NX_SUCCESS)
194         {
195
196             /* Yes it was found. */
197             break;
198         }
199
200         /* Move to the next index. */
201         i++;
202     }
203
204     /* Determine if a not found condition is present. */
205     if (status != NX_SUCCESS)
206     {
207
208         printf(" NO SUCH NAME!\n");
209
210         /* The object was not found - return an error. */
211         return(NX_SNMP_ERROR_NOSUCHNAME);
212     }
213
214     /* Determine if the entry has a get function. */
215     if (mib2_mib[i].object_get_callback)
216     {
217
218         /* Yes, call the get function. */
219         status =
220             (mib2_mib[i].object_get_callback)
221             (mib2_mib[i].object_value_ptr, object_data);
222     }
223     else
224     {
225
226         printf(" NO GET FUNCTION!");
227
228         /* No get function, return no access. */
229         status = NX_SNMP_ERROR_NOACCESS;
230     }
231
232     printf("\n");
233
234     /* Return the status. */
235     return(status);
236 }
237
238 /* Define the application's GETNEXT processing routine. */
239

```



```

240  UINT mib2_getnext_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
241  NX_SNMP_OBJECT_DATA *object_data)
242  {
243
244      UINT    i;
245      UINT    status;
246
247
248      printf("SNMP Manager GETNEXT Request For: %s", object_requested);
249
250      /* Loop through the sample MIB to see if we have information for the
251         supplied variable. */
252      i = 0;
253      status = NX_SNMP_ERROR;
254      while (mib2_mib[i].object_name)
255      {
256
257          /* See if we have found the next entry. */
258          status = nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);
259
260          /* Is the next entry the mib greater? */
261          if (status == NX_SNMP_NEXT_ENTRY)
262          {
263
264              /* Yes it was found. */
265              break;
266          }
267
268          /* Move to the next index. */
269          i++;
270      }
271
272      /* Determine if a not found condition is present. */
273      if (status != NX_SNMP_NEXT_ENTRY)
274      {
275
276          printf(" NO SUCH NAME!\n");
277
278          /* The object was not found - return an error. */
279          return(NX_SNMP_ERROR_NOSUCHNAME);
280      }
281
282
283      /* Copy the new name into the object. */
284      nx_snmp_object_copy(mib2_mib[i].object_name, object_requested);
285
286      printf(" Next Name is: %s", object_requested);
287
288      /* Determine if the entry has a get function. */
289      if (mib2_mib[i].object_get_callback)
290      {
291
292          /* Yes, call the get function. */
293          status =
294              (mib2_mib[i].object_get_callback)
295              (mib2_mib[i].object_value_ptr, object_data);
296
297          /* Determine if the object data indicates an end-of-mib condition. */
298          if (object_data -> nx_snmp_object_data_type == NX_SNMP_END_OF_MIB_VIEW)
299          {
300
301              /* Copy the name supplied in the mib table. */
302              nx_snmp_object_copy(mib2_mib[i].object_value_ptr, object_requested);
303          }
304      }
305      else
306      {
307
308          printf(" NO GET FUNCTION!");
309
310          /* No get function, return no access. */
311          status = NX_SNMP_ERROR_NOACCESS;

```

```

311     }
312
313     printf("\n");
314
315     /* Return the status. */
316     return(status);
317 }
318
319
320 /* Define the application's SET processing routine. */
321
322 UINT mib2_set_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *object_requested,
323 NX_SNMP_OBJECT_DATA *object_data)
324 {
325
326     UINT    i;
327     UINT    status;
328
329
330     printf("SNMP Manager SET Request For: %s", object_requested);
331
332     /* Loop through the sample MIB to see if we have information for the
333        supplied variable. */
334     i = 0;
335     status = NX_SNMP_ERROR;
336     while (mib2_mib[i].object_name)
337     {
338
339         /* See if we have found the matching entry. */
340         status = nx_snmp_object_compare(object_requested, mib2_mib[i].object_name);
341
342         /* Was it found? */
343         if (status == NX_SUCCESS)
344         {
345
346             /* Yes it was found. */
347             break;
348         }
349
350         /* Move to the next index. */
351         i++;
352     }
353
354     /* Determine if a not found condition is present. */
355     if (status != NX_SUCCESS)
356     {
357
358         printf(" NO SUCH NAME!\n");
359
360         /* The object was not found - return an error. */
361         return(NX_SNMP_ERROR_NOSUCHNAME);
362     }
363
364
365     /* Determine if the entry has a set function. */
366     if (mib2_mib[i].object_set_callback)
367     {
368
369         /* Yes, call the set function. */
370         status =
371             (mib2_mib[i].object_set_callback)
372             (mib2_mib[i].object_value_ptr, object_data);
373     }
374     else
375     {
376
377         printf(" NO SET FUNCTION!");
378
379         /* No get function, return no access. */
380         status = NX_SNMP_ERROR_NOACCESS;
381     }

```

```

382     printf("\n");
383
384     /* Return the status. */
385     return(status);
386 }
387
388
389 /* Define the application's authentication routine. */
390
391 UINT  mib2_username_processing(NX_SNMP_AGENT *agent_ptr, UCHAR *username)
392 {
393
394     printf("Username is:  %s\n", username);
395
396     /* Update MIB-2 objects. In this example, it is only the SNMP objects. */
397     mib2_variable_update(&ip_0, &my_agent);
398
399     /* No authentication is done, just return success! */
400     return(NX_SUCCESS);
401 }
402
403
404 /* Define the application's update routine. */
405
406 VOID  mib2_variable_update(NX_IP *ip_ptr, NX_SNMP_AGENT *agent_ptr)
407 {
408
409     /* Update the snmp parameters. */
410     snmpInPkts =          agent_ptr -> nx_snmp_agent_packets_received;
411     snmpOutPkts =         agent_ptr -> nx_snmp_agent_packets_sent;
412     snmpInBadVersions =   agent_ptr -> nx_snmp_agent_invalid_version;
413     snmpInBadCommunityNames = agent_ptr -> nx_snmp_agent_authentication_errors;
414     snmpInBadCommunityUsers = agent_ptr -> nx_snmp_agent_username_errors;
415     snmpInASNParseErrs =  agent_ptr -> nx_snmp_agent_internal_errors;
416     snmpInTotalReqVars =  agent_ptr -> nx_snmp_agent_total_get_variables;
417     snmpInTotalSetVars =  agent_ptr -> nx_snmp_agent_total_set_variables;
418     snmpInGetRequests =   agent_ptr -> nx_snmp_agent_get_requests;
419     snmpInGetNexts =      agent_ptr -> nx_snmp_agent_getnext_requests;
420     snmpInSetRequests =   agent_ptr -> nx_snmp_agent_set_requests;
421     snmpOutTooBig =       agent_ptr -> nx_snmp_agent_too_big_errors;
422     snmpOutNoSuchNames =  agent_ptr -> nx_snmp_agent_no_such_name_errors;
423     snmpOutBadValues =    agent_ptr -> nx_snmp_agent_bad_value_errors;
424     snmpOutGenErrs =      agent_ptr -> nx_snmp_agent_general_errors;
425     snmpOutTraps =        agent_ptr -> nx_snmp_agent_traps_sent;
426 }

```

Figure 1.1 Example of SNMP Agent use with NetX

Configuration Options

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

Define	Meaning
NX_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 <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .
NX_SNMP_AGENT_TIMEOUT	Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 100, but can be redefined prior to inclusion of

nx_snmp.h.

NX_SNMP_MAX_OCTET_STRING	Specifies the maximum number of bytes allowed in an octet string in the SNMP Agent. The default value is set to 255, but can be redefined prior to inclusion of <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .
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 inclusion of <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .
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 <i>nx_snmp.h</i> .

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 <i>nx_snmp.h</i> .
NX_SNMP_DISABLE_V1	Defined, this removes all the SNMP Version 1 processing in <i>nx_snmp.c</i> .
NX_SNMP_DISABLE_V2	Defined, this removes all the SNMP Version 2 processing in <i>nx_snmp.c</i> .
NX_SNMP_DISABLE_V3	Defined, this removes all the SNMP Version 3 processing in <i>nx_snmp.c</i> .

Chapter 3

Description of SNMP Agent Services

This chapter contains a description of all NetX SNMP Agent services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

`nx_snmp_agent_authenticate_key_use`
Specify authentication key (SNMP v3 only)

`nx_snmp_agent_community_get`
Retrieve community name

`nx_snmp_agent_context_engine_set`
Set context engine (SNMP v3 only)

`nx_snmp_agent_context_name_set`
Set context name (SNMP v3 only)

`nx_snmp_agent_create`
Create SNMP agent

`nx_snmp_agent_delete`
Delete SNMP agent

`nx_snmp_agent_md5_key_create`
Create md5 key (SNMP v3 only)

`nx_snmp_agent_privacy_key_use`
Specify encryption key (SNMP v3 only)

`nx_snmp_agent_sha_key_create`
Create sha key (SNMP v3 only)

`nx_snmp_agent_start`
Start SNMP agent

`nx_snmp_agent_stop`
Stop SNMP agent

`nx_snmp_agent_trap_send`
Send SNMP v1 trap

`nx_snmp_agent_trapv2_send`
Send SNMP v2 trap

`nx_snmp_agent_trapv2_oid_send`
Send SNMP v2 trap specifying OID

`nx_snmp_agent_trapv3_send`
Send SNMP v3 trap

`nx_snmp_agent_trapv3_oid_send`
Send SNMP v3 trap specifying OID

`nx_snmp_object_compare`
Compare two objects

`nx_snmp_object_copy`
Copy an object

`nx_snmp_object_counter_get`
Get counter object

`nx_snmp_object_counter_set`
Set counter object

`nx_snmp_object_counter64_get`
Get 64-bit counter object

`nx_snmp_object_counter64_set`
Set 64-bit counter object

`nx_snmp_object_end_of_mib`
Set end-of-mib value

`nx_snmp_object_gauge_get`
Get gauge object

`nx_snmp_object_gauge_set`
Set gauge object

nx_snmp_object_id_get
Get object id

nx_snmp_object_id_set
Set object id

nx_snmp_object_integer_get
Get integer object

nx_snmp_object_integer_set
Set integer object

nx_snmp_object_ip_address_get
Get IP address object

nx_snmp_object_ip_address_set
Set IP address object

nx_snmp_object_no_instance
Set no-instance value

nx_snmp_object_not_found
Set not-found value

nx_snmp_object_octet_string_get
Get octet string object

nx_snmp_object_octet_string_set
Set octet string object

nx_snmp_object_string_get
Get ASCII string object

nx_snmp_object_string_set
Set ASCII string object

nx_snmp_object_timetics_get
Get timetics object

nx_snmp_object_timetics_set
Set timetics object

nx_snmp_agent_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_trapv2_oid_send,
nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send,
nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send`,
`nx_snmp_agent_trapv3_send`, `nx_snmp_agent_trapv3_oid_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_trapv2_oid_send`,
`nx_snmp_agent_trapv3_send`, `nx_snmp_agent_trapv3_oid_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_trapv2_oid_send,
 nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send,
 nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_send

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_trapv2_oid_send,
nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send,
nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send,
nx_snmp_agent_trapv3_send, nx_snmp_agent_trapv3_oid_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_trapv2_oid_send, nx_snmp_agent_trapv3_send,
 nx_snmp_agent_trapv3_oid_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_trapv2_oid_send, nx_snmp_agent_trapv3_send,
 nx_snmp_agent_trapv3_oid_send

nx_snmp_agent_trap_send

Send SNMP v1 trap

Prototype

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

Description

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

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.												
ip_address	IP address of SNMP Manager.												
enterprise	Enterprise object ID string (sysObjectID).												
trap_type	Type of trap requested, as follows: <table data-bbox="617 1155 1341 1375"> <tr><td>NX_SNMP_TRAP_COLDSTART</td><td>(0)</td></tr> <tr><td>NX_SNMP_TRAP_WARMSTART</td><td>(1)</td></tr> <tr><td>NX_SNMP_TRAP_LINKDOWN</td><td>(2)</td></tr> <tr><td>NX_SNMP_TRAP_LINKUP</td><td>(3)</td></tr> <tr><td>NX_SNMP_TRAP_AUTHENTICATE_FAILURE</td><td>(4)</td></tr> <tr><td>NX_SNMP_TRAP_EGPNEIGHBORLOSS</td><td>(5)</td></tr> </table>	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)
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

NX_IP_ADDRESS_ERROR (0x21)	parameter pointer.
NX_OPTION_ERROR (0x0a)	Invalid destination IP address.
	Invalid parameter.

Allowed From

Initialization, Threads

Example

```

NX_SNMP_TRAP_OBJECT trap_list[5];

/* 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, IP_ADDRESS(193,2,2,61),
                                "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_trapv2_send,
 nx_snmp_agent_trapv2_oid_send, nx_snmp_agent_trapv3_send,
 nx_snmp_agent_trapv3_oid_send

nx_snmp_agent_trapv2_send

Send SNMP v2 trap

Prototype

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

Description

This service send 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 address of SNMP Manager.												
community	Community name (username).												
trap_type	Type of trap requested, as follows: <table data-bbox="617 1123 1341 1354"> <tr><td>NX_SNMP_TRAP_COLDSTART</td><td>(0)</td></tr> <tr><td>NX_SNMP_TRAP_WARMSTART</td><td>(1)</td></tr> <tr><td>NX_SNMP_TRAP_LINKDOWN</td><td>(2)</td></tr> <tr><td>NX_SNMP_TRAP_LINKUP</td><td>(3)</td></tr> <tr><td>NX_SNMP_TRAP_AUTHENTICATE_FAILURE</td><td>(4)</td></tr> <tr><td>NX_SNMP_TRAP_EGPNEIGHBORLOSS</td><td>(5)</td></tr> </table>	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)
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];

/* 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, IP_ADDRESS(193,2,2,61), "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_oid_send, nx_snmp_agent_trapv3_send,
 nx_snmp_agent_trapv3_oid_send

nx_snmp_agent_trapv2_oid_send

Send SNMP v2 trap specifying OID directly

Prototype

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

Description

This service sends an SNMP v2 trap to the SNMP Manager at the specified IP address and allows the caller to specify the OID directly.

Input Parameters

agent_ptr	Pointer to SNMP Agent control block.
ip_address	IP address of SNMP Manager.
community	Community name (username).
oid	Pointer to buffer containing OID buffer:
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];

/* Set the OID here. */
UCHAR *oid = "0.9.9.9.9.9.9.9";

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

/* Send trap to SNMP manager at 193.2.2.61. */
Status = nx_snmp_agent_trapv2_oid_send(&my_agent, IP_ADDRESS(193,2,2,61),
    "public",oid, 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_oid_send

nx_snmp_agent_trapv3_send

Send SNMP v3 trap

Prototype

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

Description

This service send an SNMP v3 trap to the SNMP Manager at the specified IP 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 address of SNMP Manager.												
community	Community name (username).												
trap_type	Type of trap requested, as follows: <table data-bbox="613 1163 1341 1386"> <tr><td>NX_SNMP_TRAP_COLDSTART</td><td>(0)</td></tr> <tr><td>NX_SNMP_TRAP_WARMSTART</td><td>(1)</td></tr> <tr><td>NX_SNMP_TRAP_LINKDOWN</td><td>(2)</td></tr> <tr><td>NX_SNMP_TRAP_LINKUP</td><td>(3)</td></tr> <tr><td>NX_SNMP_TRAP_AUTHENTICATE_FAILURE</td><td>(4)</td></tr> <tr><td>NX_SNMP_TRAP_EGPNEIGHBORLOSS</td><td>(5)</td></tr> </table>	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)
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];

/* 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, IP_ADDRESS(193,2,2,61), "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_trapv2_oid_send,
 nx_snmp_agent_trapv3_oid_send

nx_snmp_agent_trapv3_oid_send

Send SNMP v3 trap specifying the OID directly

Prototype

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

Description

This service send an SNMP v3 trap to the SNMP Manager at the specified IP address with the specified OID. This trap is basically the same as the SNMP v2 “oid” 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 address of SNMP Manager.
community	Community name (username).
oid	Pointer to buffer specifying the OID:
elapsed_time	Time system has been up (sysUpTime).
object_list_ptr	Array of objects and their associated values to be included in the SNMP trap. The list is NX_NULL terminated.

Return Values

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

Allowed From

Initialization, Threads

Example

```

NX_SNMP_TRAP_OBJECT trap_list[5];

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

/* Set the OID here. */
UCHAR *oid = "0.9.9.9.9.9.9.9";

/* Send trap to SNMP manager at 193.2.2.61. */
Status = nx_snmp_agent_trapv3_oid_send(&my_agent, IP_ADDRESS(193,2,2,61),
    "public",oid, 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_trapv2_oid_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 been 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 been 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 been 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 been 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 been 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 been successfully retrieved.
NX_PTR_ERROR	(0x16)	Invalid parameter pointer(s).

Allowed From

Initialization, Threads

Example

```
/* Get the value of ifSpeed (1.3.6.1.2.1.2.1.5.0) 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 been 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 been 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 been 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, ifAdminStatus 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

Prototype

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

Description

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

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	The IP address object has been successfully retrieved.
NX_PTR_ERROR	(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_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 IP address object

Prototype

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

Description

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

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	The IP address object has been 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_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 been 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 been 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 been 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 been 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