Renesas Synergy™ Platform

NetX™ and NetX Duo™ SNMP Agent Module Guide

R11AN0346EU0100
Rev.1.00
Oct 24, 2018

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application, and write code using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available on the Renesas Synergy™ Knowledge Base (as described in the References section at the end of this document) and should be valuable resources for creating more complex designs.
The Simple Network Management Protocol (SNMP) is an Internet Standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior.

There are three versions of SNMP, version 1, version 2, and version 3. Version 1 has been largely replaced by version 2. Version 3 is the same as version 2 for the most part except for one important difference. It supports authentication and encryption, so it is SNMP with security built in.

Note: Except where noted, NetX Duo SNMP Agent is identical to the NetX SNMP Agent. For setting up the IP instance for IPv6 in NetX Duo, see the *NetX Duo User Guide* for the Renesas Synergy™ Platform.

This document provides an overview of the key elements related to the NetX Duo SNMP Agent implementation on the Renesas Synergy Platform. The primary focus of this document is on the addition and configuration of the NetX Duo SNMP module to a Renesas Synergy Platform project. For more details on the operation of this module, consult the *NetX Duo™ SNMP Agent User Guide for the Renesas Synergy™ Platform* document. This document is available as part of an X-Ware™ and NetX™ Component Documents for Renesas Synergy™ zip file from the Renesas Synergy Gallery (https://synergygallery.renesas.com/ssp/support#read).

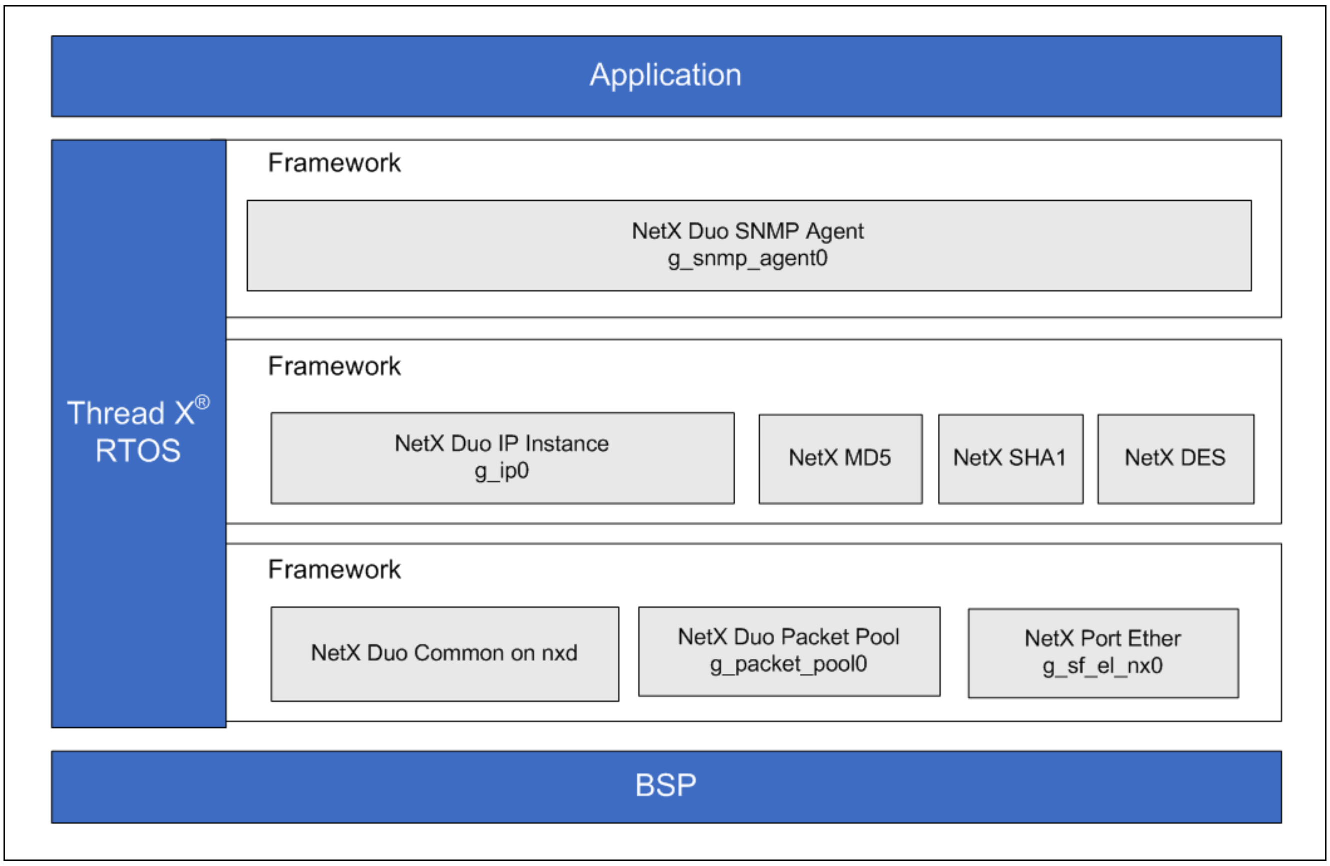


Figure 1 NetX Duo SNMP Agent Organization, Options, and Stack Implementations

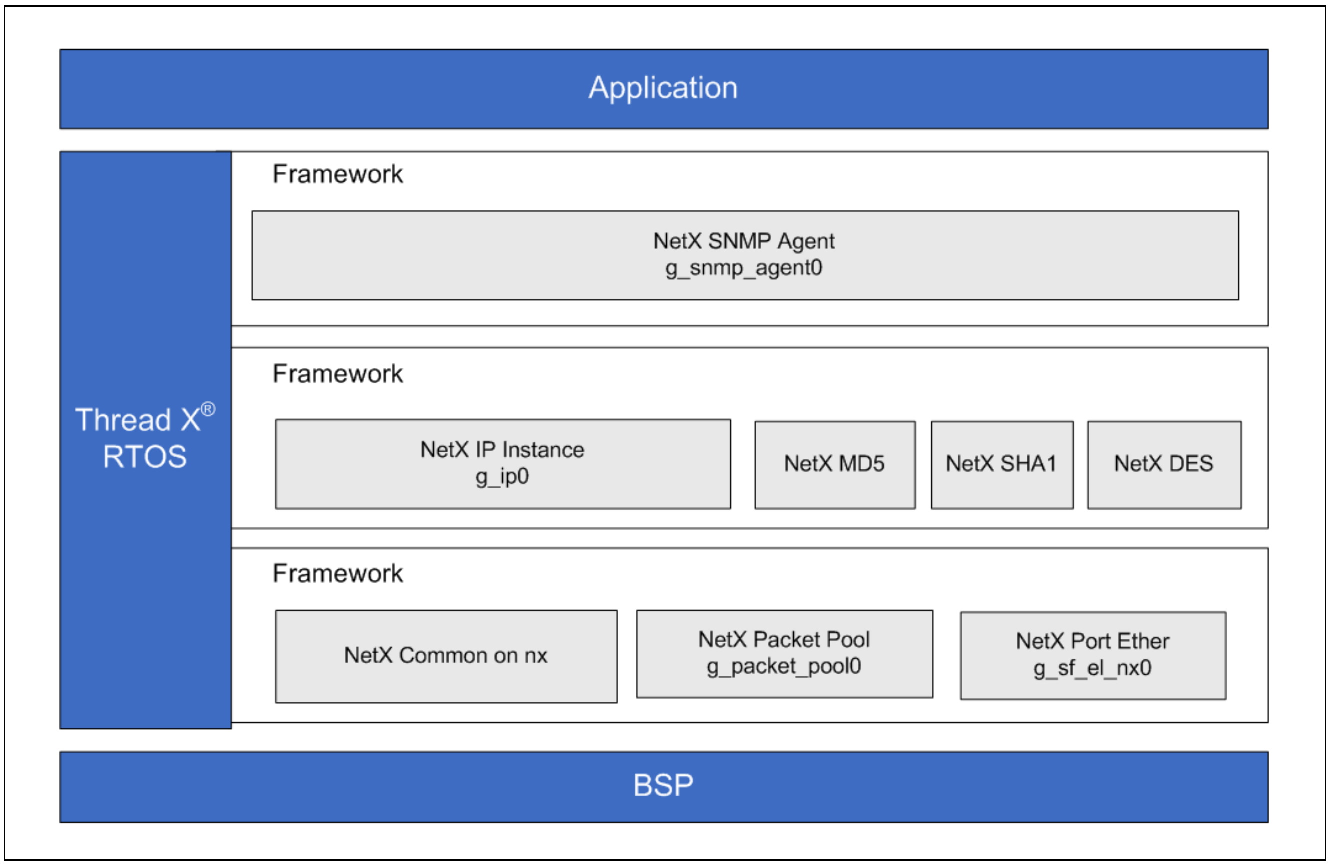


Figure 2 NetX SNMP Agent Organization, Options, and Stack Implementations

This document is divided into the following sections which can be read independently (by a more experienced Synergy developer) or in series (by developers new to the Synergy Platform).

Contents

1. NetX Duo SNMP Agent APIs Overview

The NetX Duo SNMP Agent defines APIs for creating and deleting the SNMP Agent instance, and getting its messages. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of return status values follows.

Table 1 NetX SNMP Agent API Summary

API Call Example	Description
nx_snmp_agent_authenticate_key_use(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key);	Register a previously created authentication key with the SNMP Agent for SNMP Agent responses.
nx_snmp_agent_auth_trap_key_use(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key);	Register a previously created authentication key with the SNMP Agent for SNMP trap messages.
nx_snmp_agent_community_get(NX_SNMP_AGENT *agent_ptr, UCHAR **community_string_ptr);	Retrieve the community string from the SNMP manager GET or GETNEXT request. The SNMP Agent should compare that to its own community string (see the nx_snmp_agent_public_string_test API description).
nx_snmp_agent_context_engine_set(NX_SNMP_AGENT *agent_ptr, UCHAR *context_engine, UINT context_engine_size);	Set the context engine ID of the SNMP Agent. Must specify the size of the ID string. Only used in SNMPv3 to identify the Agent to the SNMP Manager.
nx_snmp_agent_context_name_set(NX_SNMP_AGENT *agent_ptr, UCHAR *context_name, UINT context_name_size);	Set the context name of the SNMP Agent. This string must be null terminated and the size of the name must be specified. This name must be known to the SNMP Manager. Only used in SNMPv3.
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));	Create the SNMP Agent and set the Agent thread task size, packet pool for transmitting SNMP messages, and user defined callbacks for handling GET, GETNEXT, SET and username requests.
nx_snmp_agent_current_version_get(NX_SNMP_AGENT *agent_ptr, UINT *version);	Obtain the current version of SNMP based on the last message received.
nx_snmp_agent_delete(NX_SNMP_AGENT *agent_ptr);	Delete the previously created SNMP agent.
nx_snmp_agent_md5_key_create(NX_SNMP_AGENT *agent_ptr, UCHAR *password, NX_SNMP_SECURITY_KEY *destination_key);	Create a key based on a supplied password and SNMP Agent context engine ID using the MD5 algorithm. This key can be used for authentication or encryption.
nx_snmp_agent_privacy_key_use(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key);	Register a previously created security key with the SNMP Agent for encrypting and decrypting SNMPv3 messages.

API Call Example	Description
nx_snmp_agent_priv_trap_key_use(NX_SNMP_AGENT *agent_ptr, NX_SNMP_SECURITY_KEY *key);	Register a previously created security key with the SNMP Agent for encrypting SNMPv3 trap messages.
nx_snmp_agent_private_string_set(NX_SNMP_AGENT *agent_ptr, UCHAR *private_string);	Register a null terminated privacy string to the SNMP Agent. Only used in SNMP1 and SNMPv2.
nx_snmp_agent_private_string_test(NX_SNMP_AGENT *agent_ptr, UCHAR *community_string, UINT *is_private);	The SNMP Agent compares the privacy string in a SET request with the its own privacy string to determine if the SET request will be permitted.
nx_snmp_agent_public_string_set(NX_SNMP_AGENT *agent_ptr, UCHAR *public_string);	Register a null terminated public string to the SNMP Agent. Only used in SNMP1 and SNMPv2.
nx_snmp_agent_public_string_test(NX_SNMP_AGENT *agent_ptr, UCHAR *community_string, UINT *is_public);	The SNMP Agent compares the public string in a GET or GETNEXT request with the its own public string to determine if the request will be permitted.
nx_snmp_agent_request_get_type_test(NX_SNMP_AGENT *agent_ptr, UINT *is_get_type);	Determine if the last SNMP packet received was a GET, GETNEXT, or GET_BULK_REQUEST request type (returns TRUE) or a SET request (returns FALSE). Intended for use in the username callback for type of request received and checking public or private strings in the request message.
nx_snmp_agent_set_interface(NX_SNMP_AGENT *agent_ptr, UINT if_index);	Determine on which network interface to run the SNMP Agent protocol. The default interface is the primary (0) interface.
nx_snmp_agent_sha_key_create(NX_SNMP_AGENT *agent_ptr, UCHAR *password, NX_SNMP_SECURITY_KEY *destination_key);	Create a key based on a supplied password and SNMP Agent context engine ID using the SHA1 algorithm. This key can be used for authentication or encryption.
nx_snmp_agent_start(NX_SNMP_AGENT *agent_ptr);	Start the SNMP Agent thread task. This task waits to receive SNMP messages and formulates the response to the SNMP Manager.
nx_snmp_agent_stop(NX_SNMP_AGENT *agent_ptr);	Stop the SNMP Agent thread task. The SNMP Agent thread can be restarted by calling the nx_snmp_agent_start API.
nx_snmp_agent_trap_send(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * community, UCHAR *enterprise, UINT trap_type, UINT trap_code, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv1. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed.
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);	Send a trap message in SNMPv2. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed.
nx_snmp_agent_trapv3_send(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv3. This does not result from a request from the SNMP Manager. The SNMP application sends out traps as needed. Both the SNMP agent and browser must previously agree on the security (authentication and encryption) settings.
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);	Send a trap message in SNMPv2. This differs from nx_snmp_agent_trapv2_send in that it allows the caller to specify the OID directly.

API Call Example	Description
nx_snmp_agent_trapv3_oid_send(NX_SNMP_AGENT *agent_ptr, ULONG ip_address, UCHAR * username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv3. This differs from nx_snmp_agent_trapv3_send in that it allows the caller to specify the OID directly.
nx_snmp_agent_v3_context_boots_set(NX_SNMP_AGENT *agent_ptr, UINT boots);	Set the number of times the SNMP Agent has rebooted since the last communication with the SNMP Manager. Used in SNMPv3 only.
nx_snmp_agent_version_set(NX_SNMP_AGENT *agent_ptr, UINT enabled_v1, UINT enable_v2, UINT enable_v3);	Determine which type of SNMP packets the SNMP Agent will process. The application can choose V1, V2 and/or V3. Packets received from which the SNMP Agent is not enabled are dropped.
****nxd_snmp_agent_trap_send**(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR *community, UCHAR *enterprise, UINT trap_type, UINT trap_code, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv1 over IPv6. Note that this takes an NXD_ADDRESS *ip_address instead of ULONG ip_address.
****nxd_snmp_agent_trapv2_send**(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR * community, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv2. Note that this takes an NXD_ADDRESS *ip_address instead of ULONG ip_address. The ip_address is the destination IP address of the trap message.
****nxd_snmp_agent_trapv3_send**(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR *username, UINT trap_type, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv3. Note that this takes an NXD_ADDRESS *ip_address instead of ULONG ip_address. The ip_address is the destination IP address of the trap message.
****nxd_snmp_agent_trapv2_oid_send**(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR *community, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv2. Note that this takes an NXD_ADDRESS *ip_address instead of ULONG ip_address. The ip_address is the destination IP address of the trap message.
****nxd_snmp_agent_trapv3_oid_send**(NX_SNMP_AGENT *agent_ptr, NXD_ADDRESS *ip_address, UCHAR * username, UCHAR *oid, ULONG elapsed_time, NX_SNMP_TRAP_OBJECT *object_list_ptr);	Send a trap message in SNMPv3. Note that this takes an NXD_ADDRESS *ip_address instead of ULONG ip_address. The ip_address is the destination IP address of the trap message.
**Not available in NetX SNMP Agent. For all API referencing a user name, see the discussion in section *Usernames in SNMP* in Section 3 for more details about community and security user names in SNMP.	

The following API calls are for processing data items into the SNMP Agent response. The snmp_mib_helper.h file in the module guide project defines the Management Information Base (MIB) database used in this project. It uses these functions for setting the value of each object in the MIB database.

API Call Example	Description
nx_snmp_object_copy(UCHAR *source_object_name, UCHAR *destination_object_name);	This copies the string pointed to by source_object_name into the destination_object_name buffer (typically a trap message).
nx_snmp_object_counter_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This extracts the counter data from the location pointed to by source_ptr into the object data. Also used internally to copy internal counters of SNMPv3 statistics into error messages in SNMPv3 messages.

API Call Example	Description
nx_snmp_object_counter_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This sets the value of data extracted from the object_data into the location pointed to by the destination_ptr.
nx_snmp_object_counter64_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This extracts the counter data from the location pointed to by source_ptr into the object data. The difference with nx_snmp_object_counter_get is the value is 64 bits instead of 32 bits in size.
nx_snmp_object_counter64_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This sets the value of data extracted from the object_data into the location pointed to by the destination_ptr. The difference with nx_snmp_object_counter_set is the value is 64 bits instead of 32 bits in size.
nx_snmp_object_compare(UCHAR *requested_object, UCHAR *reference_object);	This compares the two input objects and if equal returns NX_SUCCESS.
nx_snmp_object_end_of_mib(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data);	This appends an END_OF_MIB_VIEW macro as the input object's value. This signals the end of the MIB. See the snmp_mib_helper.h for an example.
nx_snmp_object_gauge_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This sets the input object to type SNMP GAUGE and places the value pointed to by the source_ptr into the object value.
nx_snmp_object_gauge_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This verifies the input object is an SNMP GAUGE data type, and extracts the value into the location pointed to by the destination pointer.
nx_snmp_object_id_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function retrieves the object ID from the specified source location and writes it into the object data value.
nx_snmp_object_id_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function retrieves the ASCII string from the input object and writes it to the area pointed to by the destination pointer.
nx_snmp_object_integer_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data)	This retrieves the object integer from the specified source location and stores it to the object data.
nx_snmp_object_integer_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the integer from the input object and places it in the destination.
nx_snmp_object_ip_address_get (VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the IP address from the specified source location and stores it to the object data.
nx_snmp_object_ip_address_set (VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the IP address from the input object and places it in the destination.
** **nx_snmp_object_ipv6_address_get** (VOID *source_ptr,	This retrieves the IPv6 address from the specified source location and stores it to the object data.

API Call Example	Description
NX_SNMP_OBJECT_DATA *object_data);	
****nx_snmp_object_ipv6_address_set** (VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the IPv6 address from the input object and places it in the destination.
nx_snmp_object_octet_string_get (VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data, UINT length);	This retrieves the string data from the specified source location and stores it to the object data.
nx_snmp_object_octet_string_set (VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the string from the input object and places it in the destination.
nx_snmp_object_no_instance(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function places a no-instance value (NX_SNMP_ANS1_NO_SUCH_INSTANCE) in the object data.
nx_snmp_object_not_found(VOID *not_used_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function places a not-found value (NX_SNMP_ANS1_NO_SUCH_OBJECT) in the object data.
nx_snmp_object_string_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function retrieves the ASCII string from the specified source location and stores it to the object data.
nx_snmp_object_string_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the ASCII string from the input object and stores it to the destination.
nx_snmp_object_timetics_get(VOID *source_ptr, NX_SNMP_OBJECT_DATA *object_data);	This function retrieves the data of type timer ticks from the specified source location and stores it to the object data.
nx_snmp_object_timetics_set(VOID *destination_ptr, NX_SNMP_OBJECT_DATA *object_data);	This retrieves the timer tick from the input object and stores it to the destination.

**Not available in NetX SNMP Agent.

Note: For more detailed descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the associated Express Logic User's Manual accessible as described in the Reference section at the end of this document.

Table 2 Error Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_PTR_ERROR*	Invalid input pointer parameter
NX_SNMP_UNSUPPORTED_AUTHENTICATION*	The authentication key is of an unknown or unsupported type (for example, not MD5 or SHA).
NX_SNMP_INVALID_PDU_ENCRYPTION*	The encryption key is of an unknown or unsupported type (for example, not MD5 or SHA).
NX_IP_ADDRESS_ERROR*	IP address supplied in a trap send API is null or invalid.

NX_SNMP_ERROR	Internal processing error, such as, not able to append data into the SNMP response.
NX_NOT_ENABLED	SNMP Agent is not enabled with the correct security settings for certain operations such as sending messages or processing authentication and encryption keys.
NX_SNMP_ERROR_TOOBIG	Data exceeds the size of the response buffer or exceed the allowable size of the parameter for example, NX_SNMP_MAX_USER_NAME.
**NX_SNMP_INVALID_IP_PROTOCOL_ERROR	An IPv6 address is received in a trap send API but the NetX Duo library is not enabled for IPv6.

Note: Internal NetX operations may return Common Error Codes. See the *SSP Reference Manual* listed in the Reference Section at the end of this document for a definition of all relevant Error codes.

*These are error codes that are only returned if error checking is enabled. See the *NetX User Guide* for the Renesas Synergy™ Platform for more details on error checking services in NetX.

**These error codes do not apply to NetX SNMP Agent.

2. NetX Duo SNMP Agent Operational Overview

The SNMP Agent module makes use of the underlying NetX/NetX Duo stack to perform operations, along with NetX MD5, NetX SHA1, and NetX DES modules for authentication and encryption in SNMP v3 operation.

The reader must be familiar with the general use of SNMP and how OID strings are constructed to describe object databases. If not, there are abundant sources online for the lengthy introduction of how SNMP works and what MIB trees are for describing data set. This document explains how NetX supports SNMP Agent and how to implement the NetX Duo SNMP Agent application.

The NetX Duo SNMP Agent requires a previously created NetX IP instance and a packet pool to send SNMP messages and traps to the SNMP manager. The packet pool can be the same one used by the IP instance or a separate packet pool. Whichever packet pool is used by the SNMP Agent, it must have a payload large enough for SNMP responses which can be quite large, particularly for responding to a GETBULK request. The recommended packet payload is the device MTU (typically 1518 bytes). Because the NetX Duo SNMP Agent utilizes UDP services, UDP must be enabled on the IP instance.

Synergy auto-generated code creates the IP instance, packet pool(s), SNMP Agent instance, and enables UDP.

The SNMP Agent module is created and the community read and write usernames (public and private, respectively) are registered to the SNMP Agent by the auto generated code, if Auto Initialization property is enabled (default is enabled). Otherwise, the application creates it with the nx_snmp_agent_create API and sets the community strings with the nx_snmp_agent_public_string_set and nx_snmp_agent_private_string_set APIs. With SNMP creation, the user defined callback functions for handling Username, GET, GETNEXT, or SET requests are registered with the SNMP Agent, and the Agent can verify the identity of the SNMP Manager with the community strings (in SNMPv2 at least). The SNMP Properties section and Explanation of the Example MIB Table section provide more detail about the handlers and how they access an MIB database.

SNMP protocol has three versions, version 1 which is virtually out of date, version 2, and version 3 which has authentication and encryption security. NetX Duo SNMP Agent supports all versions. By default, it is enabled for all three so it can respond to SNMPv1, v2, or v3 queries. To modify this, the application can call the nx_snmp_agent_version_set API to enable or disable each protocol separately.

SNMP also requires an MIB database that associates object data with user defined functions for accessing or setting the specified data. The SNMP project in this module guide comes with a simple MIB database in the snmp_mib_helper.h file which the developer can use as is or customize for their own MIB database.

The application thread entry function need only wait for the network link to be enabled using the nx_ip_status_check API. SNMP set up is complete, assuming SNMPv3 security is not required. SNMPv3 security is described in more detail in the SNMPv3 Security section below. After the network is enabled, the SNMP Agent can be started and start listening for and responding to SNMP Manager queries. The application can send trap messages directly to the SNMP Manager as needed (no request from the SNMP Manager required).

2.1 SNMP Data Types

IPv4 addresses are handled as ULONG data types. IPv6 addresses are, however, handled as octet strings. This is because of the complexity of the IPv6 128 addressing with the optional '::' formatting to eliminate one or more zero bytes. Formatting IPv6 octet strings into the full 128 bits array is left to the application callback functions.

SNMP Agent supports 64 data types (counter64 data type) and long form integers. A long form integer is where the length of the integer cannot be contained in 1 byte. These are relatively rare. More information about Type Length Variable (TLV) processing of integers in ASN1 BER can be found in most SNMP text books or numerous sources online (https://en.wikipedia.org/wiki/X.690#BER_encoding).

Octet strings must have a defined length in the MIB table (stored in the length table of the module guide project in snmp_mib_helper.h). Unlike string types which are delimited by the null (0x0) character, octet strings are not delimited. Therefore, the callback functions for setting and getting data into and out of the MIB table respectively save the current data size. This will be explained in more specific detail in the "Explanation of the Example MIB Table" section elsewhere in this document.

2.2 SNMPv3 Security

SNMPv3 by itself is a complex subject. Entire books have been written on the subject. This document assumes that the reader has a basic understanding of authentication and encryption (sometimes referred to as privacy) security settings in SNMPv3.

The SNMP Agent needs additional setup compared with SNMPv1 or SNMPv2. It needs to set the engine boot count, (the agent is referred to as the 'engine' and boot means number of times rebooted since its last SNMP session). It also needs to set the engine context ID, which is an octet string of user defined size.

To enable authentication, the application creates an authentication key using either the MD5 or SHA key type. All keys require a password. If the key is successfully created, it must be registered to the SNMP Agent.

If privacy security is desired, the application must create privacy key, using either the MD5 or SHA key type. All keys require a password. If the key is successfully created, it must be registered to the SNMP Agent.

Note: Privacy security requires authentication security to be enabled, while authentication does not require privacy security to be enabled.

Traps in SNMPv3 can also enable authentication and privacy and require yet another set of keys. Traps need not have the same security as the SNMP Agent responses. For example, the SNMP Agent may require authentication of SNMPv3 requests and responses, while SNMPv3 traps may be sent with no security.

When the SNMP Manager wishes to contact the SNMP Agent using SNMPv3, there is a discovery protocol where it sends a simple query with all encryption and authentication parameters in the SNMP v3 header set to NULL. The SNMP Agent sends a report with only its context engine ID set. The Manager responds with a getnext query with the boot count and engine boot time set, plus its user name, and its authentication and privacy parameters, if such security is enabled, filled in. The SNMP Agent then confirms the handshake with a response message and its own engine ID, boot count, and boot time filled in and, if enabled, the authentication and privacy parameters. At this point, the SNMP Agent is ready to handle Manager requests and the Manager will accept SNMPv3 traps from the SNMP Agent.

Note: The Manager and Agent security settings must match. If one uses only Authentication, the other can only use Authentication. And they must use the same key type and same password for computing the security parameters.

Even if this is all handled, this handshake can sometimes be tricky depending on the quality of the SNMP Manager. Most of them are not expecting the SNMP Agent to be in a development process, so they expect a sequential increase of the boot count with each SNMP session, and accurate update of engine boot time (within 150 seconds of what the SNMP Manager expects to be the boot time). This is not always practical for testing and debugging the SNMP Agent applications. Depending on the SNMP Manager tool, sometimes the easiest solution is to restart both Agent and Manager so that time and boot count is reset to zero.

2.3 Usernames in SNMP

SNMPv1 and SNMPv2: For getting (reading) data from the MIB table, the SNMP Agent/Browser have a shared community string, which defaults to “public” in this module guide project. For setting (writing) data into the MIB table, the SNMP Agent/Browser have a shared private string, which defaults to “private” in this module guide project. For trap messages, the third input, UCHAR *community should be the same as the public (community) string registered with the SNMP Agent.

Public and private strings can be set by calling the nx_snmp_agent_private_string_set and nx_snmp_agent_public_string_set API.

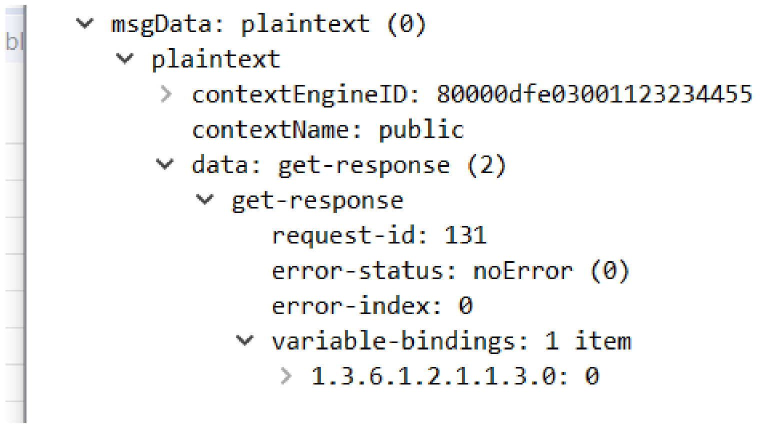
SNMPv3: When the SNMP Agent receives a request from the SNMP Manager, internally it calls its snmp_agent_username_process callback as it processes the SNMP request. This callback should compare the security name received from the SNMP Manager/browser against its own security name. If they don’t match, the username callback should return an error. The API for sending trap messages has a username input as the third input, UCHAR *username. This should be set to the SNMP security name. Currently there is no direct way to access the SNMP Agent security name. In the module guide project, the nxd_snmp_agent_trapv3_oid_send accesses the security name directly for the UCHAR *username input:

```
nx_snmp_agent_trapv3_send(snmp_ptr, mib_address,
                          (UCHAR *)&snmp_ptr -> nx_snmp_agent_v3_security_user_name[0],
                          SNMP_TRAP_COLDSTART, temp, &trap_list[0]);
```

2.4 Error Reports

Error reports are sent out by the SNMP Agent if the SNMPv3 handshake fails.

Errors in SNMPv2 or SNMPv3 in processing data received from the SNMP Manager, or creating the response to the query result in the SNMP Agent response containing the error information added to the object list in the message body with a non-zero error-index, and error-status indicating error type. Normally, error-index is zero and error-status is “noError” as shown in the packet trace detail below.



```
v msgData: plaintext (0)
  v plaintext
    > contextEngineID: 80000dfe03001123234455
      contextName: public
    v data: get-response (2)
      v get-response
          request-id: 131
          error-status: noError (0)
          error-index: 0
        v variable-bindings: 1 item
          > 1.3.6.1.2.1.1.3.0: 0
```

2.5 SNMP Manager

This project was developed using MG-Soft for the reasonably priced, not too complex SNMP Manager tool. There are other SNMP Manager tools but some are quite complex (industrial level) and are 70 MB or more in size. SNMP Manager ‘freeware’ do not always support SNMPv3 protocol to the letter.

Set up the Manager to use the same authentication (enabled/disabled, and same key type MD5 or SHA). Same is the case with encryption, if enabled. For Trap security, these require a separate set of keys. It is recommended to use the same key type for traps. Viewers discretion is recommended for using same passwords.

2.6 NetX Duo SNMP Agent Important Operational Notes and Limitations

NetX Duo SNMP Agent features are as follows:

- The NetX Duo SNMP Agent is compliant with RFC1155, RFC1157, RFC1215, RFC1901, RFC1905, RFC1906, RFC1907, RFC1908, RFC2571, RFC2572, RFC2574, RFC2575, RFC 3414, and related RFCs. RFC 3414 defines SNMPv3.
- The SNMP Agent does not have Datagram Transport Layer Security (DTLS) support.
- The NetX Duo SNMP Agent supports SNMP version 1, 2, and 3. The SNMPv3 implementation supports MD5 and Secure Hash Algorithm 1 (SHA-1) authentication, and Data Encryption Standard (DES) encryption.
- Provides a mechanism to register callbacks for handling Get Username, GET, SET, GETNEXT requests when creating an SNMP agent.

A Few Comments about SNMP Agent Properties

Section 6 has more details on the recommended settings of the SNMP Agent. However, there are a few properties which require some discussion.

- **UDP port number -** The SNMP Agent socket is bound to the port defined in this property. By default, it is set to 161. To send traps, it should send the Trap Destination Port. By default, this is set to 162.
- **SNMP Version [x] -** To disable any of the SNMP versions, set the SNMP Version [x] properties accordingly. At run time, enabling or disabling versions is accomplished by calling the nx_snmp_agent_version_set API.
- **Minimum SNMP packet size -** This is somewhat misnamed in that it allows the SNMP Agent to specify to the SNMP Manager the maximum size SNMPv3 message it can send or receive. All SNMPv3 implementations must support at least 484 bytes. This should be based on the packet pool payload used by the SNMP Agent. If the payload is 1518 bytes, there should be 1518 bytes – IP header – UDP header for SNMP message including the header, or 1518 – 54 = 1474 bytes.
- **Max User Name Size -** This is the limit on the size of the public string, also referred to as the community string. Used in SNMPv2 for how the SNMP Manger identifies itself to the SNMP Agent.

For SNMPv3 only

- **Max trap Key Size -** Size of trap security key. There is usually no reason to change the default value of 64.
- **Max security Key Size -** Size of SNMP agent security key. There is usually no reason to change the default value of 64.
- **Max trap Name Size -** This property is not in use.
- **Max context string size -** This is the size limit on the context engine ID, which is part of the Global data in the SNMPv3 header.
- **Interval between SNMP packet processing** (timer ticks) - This property is not in use.

Common properties

- **Read Community String -** Set the public string (defaults to “public”) and use to verify the SNMP Manager Get requests to read from MIB data.
- **Write Community String** - Set the private string (defaults to “private”) and use to verify the SNMP Manager Set requests to write to MIB data.
- **SNMP agent instance id -** This property is not in use.
- **Name of generated initialization function -** This defaults to snmp_agent_init0 and is called if Auto Initialization is enabled. It can be replaced with a user defined function. Alternatively, if Auto Initialization is disabled, the creation of the SNMP Agent must be done by the application.
- **Auto Initialization -** If this is set to disabled, the application must create the NetX Duo SNMP Agent, and set the public and private strings.
- **Handler functions -** Each type of query for data in the MIB database requires user-defined callback function. For the GET Request handler, the callback function takes as input a pointer to the SNMP agent, an object requested (a string comprising the OID of the object), and an object into which to store the data. These callback functions are defined in the list of API in Section 2. A more detailed explanation of how the handlers work is provided in the Explanation of the Example MIB Table section below.

2.6.1 Explanation of the Example MIB Table

The module guide comes with a small MIB defined in the snmp_mib_helper.h file. It defines the MIB table structure, and populates it with entries, each with a unique OID and get/set function as appropriate. The Get, GetNext, and Set callback functions assigned to the SNMP agent locate the OID in the table, and invoke the Get or Set function depending on whether the SNMP query type is GET or SET. The SNMP Agent Get/Set function, in turn, retrieves or sets the data in the MIB database. An example is as follows:

1. The SNMP Agent thread task receives a GET request from the SNMP Manager.
2. It calls the GET request callback registered to it such as, mib_get_processing. The application must define this function as follows:

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

   This is the general structure of the example MIB table in snmp_mib_helper.h:

```
typedef struct MIB_ENTRY_STRUCT
{
    UCHAR        *object_name;
    void         *object_value_ptr;
    UINT         (*object_get_callback)(VOID *source_ptr,
                                        NX_SNMP_OBJECT_DATA *object_data);
    UINT         (*object_set_callback)(VOID *destination_ptr,
                                        NX_SNMP_OBJECT_DATA *object_data);
} MIB_ENTRY;
```

   Following is a typical entry in the MIB table:

```
{(UCHAR *) "1.3.6.1.2.1.3.1.1.2.0",
         &atPhysAddress[0],
         nx_snmp_object_octet_string_get,
         nx_snmp_object_octet_string_set,
         0},
```

   The first field is the object name, which in SNMP parlance is the OID string. The second field is the value of the OID, which is also defined in the snmp_mib_helper.h:

```
UCHAR   atPhysAddress[] = {0x00,0x04,0xac,0xe3,0x1d,0xc5};
                          /* atPhysAddress:OctetString RW */
```

   The next two fields are the get and set handler function pointers. In this case the data is read/write, and the caller should use the nx_snmp_object_octet_string_get and nx_snmp_object_octet_string_set to retrieve data from and write data into the MIB database respectively.

3. The mib2_get_processing() performs the following logic for a GET query:
    A. Go through the list of MIB objects in the mib2_mib table.
    B. Compare the UCHAR *object_requested with each object name (also an OID string) in the mib2_mib table [for example, object_name = "1.3.6.1.2.1.3.1.1.2.0"].
    C. If not found, return a non-zero error status to the caller (SNMP Agent thread task). The SNMP Agent will then respond by sending the SNMP Manager an error message.
    D. If found, determine if there is a read function for that object in the mib2_mib table (mib2_mib[i].object_get_callback is nx_snmp_object_octet_string_get() in this case).
    E. Call the SNMP callback function for reading an octet string, nx_snmp_object_string_get, with a pointer to the data source, in this case the value of the object [object_value_ptr] and a pointer to the destination object to store it to:

```
status = (mib2_mib[i].object_get_callback)(mib2_mib[i].object_value_ptr,
                                          object_data);
```

    The above line corresponds to this API call:

```
nx_snmp_object_octet_string_get(mib2_mib[i].object_value_ptr,
                                object_data);
```

    F. Return success or failure if able to perform the GET. If successful, the SNMP Agent sends a response message with no errors to the SNMP Manager.

The only difference for a SET request is that before calling the write function for octet strings, nx_snmp_object_octet_string_set, the set handler (set_mib_processing) applies the length of the octet string (if the object is an OCTET STRING type) to the object:

```
        if (object_data -> nx_snmp_object_data_type ==
NX_SNMP_ANS1_OCTET_STRING)
             {
                  mib2_mib[i].length = object_data ->
nx_snmp_object_octet_string_size;
             }
```

Then it calls the write function

```
status = (mib2_mib[i].object_set_callback)(mib2_mib[i].object_value_ptr,
                                          object_data);
```

The reference to "mib2" in the user-defined handler names comes from the RFC 1213 specification, which redefined the Management Information Base for SNMP to MIB-II.

2.7 NetX Duo SNMP Agent Limitations

This version of the NetX Duo SNMP Agent has the following constraints:

- No support for RMON
- SNMP v3 Inform messages are not supported
- SNMP Agent does not support OPAQUE or NSAP data types
- There is no IPv6 data type yet in SNMP. IPv6 addresses are taken as is in the octet string data type. Formatting IPv6 octet strings into the full 128 bits array is left to the application callback functions.

3. Including the NetX Duo SNMP Agent in an Application

This section describes how to include the NetX Duo SNMP Agent in an application using the SSP configurator.

Note: This section assumes that you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters in the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP based applications.

To add the NetX Duo SNMP Agent to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the NetX Duo SNMP Agent is g_snmp_agent0 and this is shown in the below table. This name can be changed in the associated Properties window).

Resource	ISDE Tab	Stacks Selection Sequence
g_snmp_agent0 NetX SNMP Agent	Threads	New Stack> X-Ware> NetX> Protocols> NetX SNMP Agent
g_snmp_agent0 NetX Duo SNMP Agent	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo SNMP Agent

When the NetX Duo SNMP Agent is added to the Thread Stack, as shown in the following figure, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any pink banded modules will bring up the "New" icon and then will show the possible choices.

The SNMP Agent needs a packet pool for its creation. Select the **Add NetX Packet Pool** with the pink band connected to the NetX Duo SNMP Agent block. Choose to use the same packet pool created for the NetX IP instance, g_packet_pool0. This is shown in the following figure. Or, choose to create a new packet pool and fill in the properties box to specify size of payload and number of packets. If choosing this option, the packet pool block directly below the SNMP Agent element in the following figure would indicate a different packet pool, g_packet_pool1 by default.

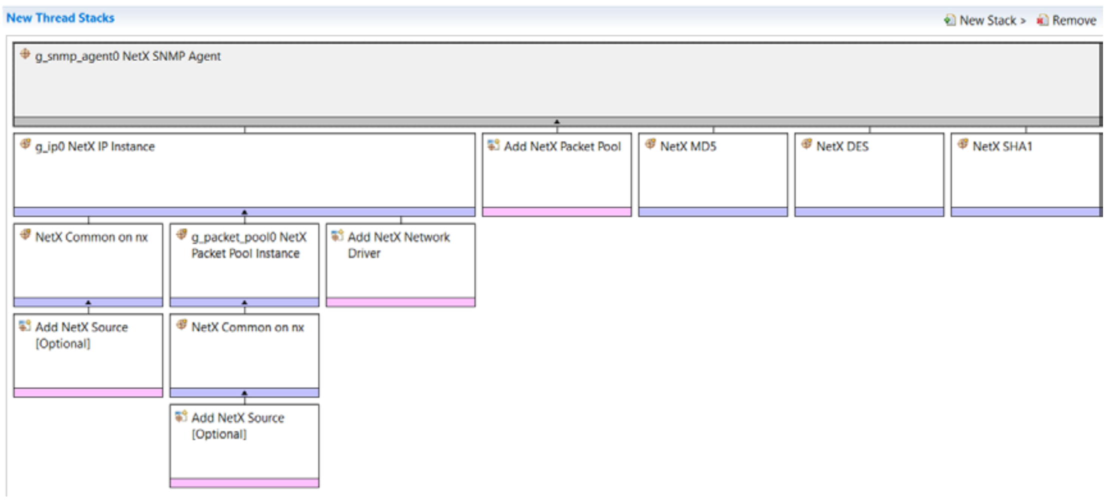


Figure 3 NetX SNMP Agent Stack

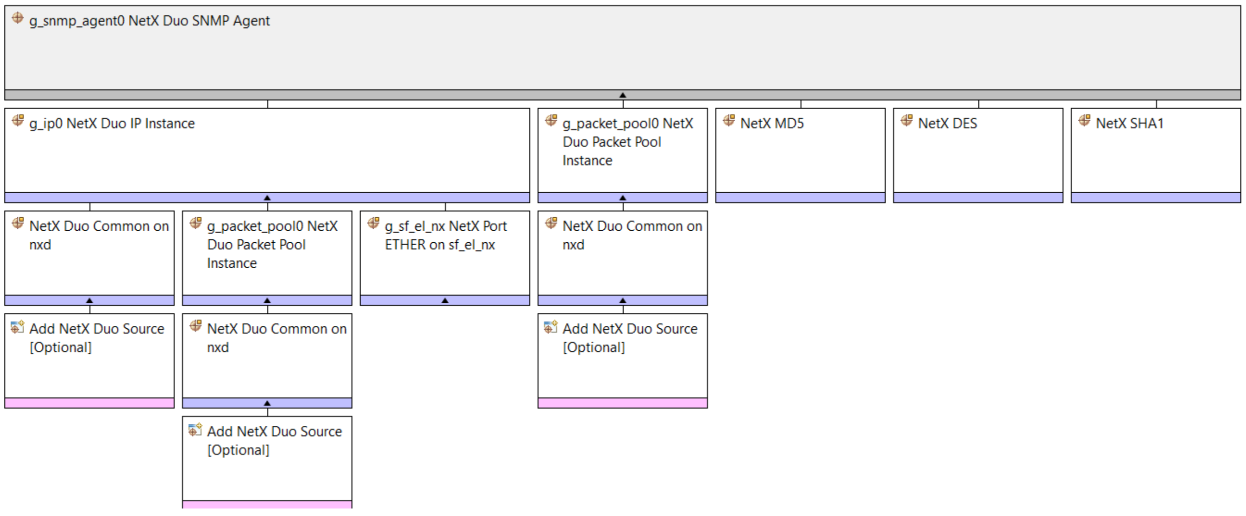

g_snmp_agent0 NetX Duo SNMP Agent
g_ip0 NetX Duo IP Instance
g_packet_pool0 NetX Duo Packet Pool Instance
NetX MD5
NetX DES
NetX SHA1
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
Add NetX Duo Source [Optional]

Figure 4 NetX Duo SNMP Agent Stack

4. Configuring the NetX Duo SNMP Agent Module

The NetX Duo SNMP Agent module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

Note: You may want to open your ISDE and create the NetX Duo SNMP Agent and explore the property settings in parallel with looking over the Configuration Table Settings that follow. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 3 Configuration Settings for NetX Duo SNMP Agent

ISDE Property	Setting	Description
Properties common to all SNMP instances in a project		
Internal thread stack size (bytes)*	4096	SNMP Agent Thread stack size. Larger than most NetX application thread stacks but this is a very large implementation.
SNMP agent priority*	3 Default: 16	SNMP Agent Thread task priority It should be set to a lower priority than the IP thread task priority. A lower priority means a higher priority number with 1 being the highest assignable priority. If the IP instance is set to 3, the SNMP Agent thread should be assigned to 4 or lower depending on the application requirements.
Type of service for SNMP responses	Normal	SNMP Agent UDP Socket service type
Fragment enable for SNMP PDU requests	Don't Fragment	Indicate to the SNMP Manager if it is ok to send fragmented packets. Usually not recommended for the high overhead of processing and potential for draining packet pools down.
SNMP Socket time to live	128	NMP socket time to live selection
Agent timeout (timer ticks)	100	Time out for internal operations such as allocating packets
Max octet string size (bytes)	255	Maximum length of an OID octet string
Max context string size (bytes)	32	Maximum length for the context engine string. This is how the SNMP agent identifies itself in SNMPv3.
Maximum User name Size (bytes)	64	Maximum length of user name. This is how the SNMP agent identifies itself in SNMPv2.

ISDE Property	Setting	Description
Max security Key Size (bytes)	64	Maximum length of either authentication or encryption key in SNMPv3
Minimum SNMP packet size (bytes)	560	This is the maximum message size advertised by packets sent by the SNMP Agent to the browser.
UDP port number	161	Socket port for the SNMP Agent socket to bind to
Trap destination port	162	Destination port for the SNMP Agent trap messages to the SNMP Manager
Max trap Name Size	NA	No longer in use
Max trap Key Size	NA	No longer in use
Interval between SNMP packet processing timer ticks	NA	No longer in use
SNMP Version 1	Enabled	SNMP agent is set to Enabled for SNMP Version 1
SNMP Version 2	Enabled	SNMP agent is set to Enabled for SNMP Version 2
SNMP Version 3	Enabled	SNMP agent is set to Enabled for SNMP Version 3
Properties specific to each instance of SNMP Agent*		
Name	g_snmp_agent0	Name of the SNMP instance
Read Community String	public	‘User name’ to identify the SNMP Manager to the SNMP Agent for requests for reading data from the MIB in SNMPv2.
Write Community String	private	‘User name’ to identify the SNMP Manager to the SNMP Agent for requests for writing data to the MIB in SNMPv2.
Name of SNMP Username* Handler	mib2_username_processing (Default: sf_snmp0_username_handler)	This is a user defined callback function pointer registered with the SNMP Agent when it is created. It is used to verify the username (or community string) in the requests received from the SNMP Manager.
Name of SNMP GET Handler*	mib2_get_processing (Default: sf_snmp0_get_handler)	This is a user defined callback function pointer registered with the SNMP Agent when it is created. It is used to find the object specified in a GET request from the SNMP Manager and fill in the data for that object from the MIB.
Name of SNMP GETNEXT Handler*	mib2_getnext_processing (Default: sf_snmp0_getnext_handler)	This is a user defined callback function pointer registered with the SNMP Agent when it is created. It is used to find the object specified in a GETNEXT request from the SNMP Manager and fill in the data for that object from the MIB. This handler is also used in the GETBULK request processing.

ISDE Property	Setting	Description
Name of SNMP SET Handler*	mib2_set_processing (Default: sf_snmp0_set_handler)	This is a user defined callback function pointer registered with the SNMP Agent when it is created. It is used to find the object specified in a SET request from the SNMP Manager and fill in the data for that object from the MIB.
Name of generated initialization function	snmp_agent_init0	If *Auto Initialization* is enabled, this function is run before the thread entry function is called. The default snmp_agent_init0 creates the SNMP Agent using the handlers described above, and the SNMP Agent thread task using the stack size and priority property settings in this table.
Auto Initialization	Enable	If enabled, the function set as the generated initialization function is called before the thread entry function. If set to disabled, the thread entry function must create the SNMP Agent.
SNMP agent instance id	NA	Not in use

*These properties will have no effect if Auto initialization is set to Disable. That is because the function that uses these parameters in the nx_snmp_agent_create API is not called.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different MAC or IP addresses. The configurable properties for the lower level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

4.1 Configuration Settings for the NetX Duo SNMP Agent Low Level Drivers

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated with the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Table 4 Configuration for the Application Thread

ISDE Property	Setting	Description
Symbol	snmp_agent_thread	Symbol name
Name	SNMP Agent App Thread	Thread name
Stack Size (bytes)	4096	Stack size for thread entry application
Priority	3	Thread priority (1 is highest)
Auto start	Enabled	If Auto start is set to Enabled, code is automatically generated that initializes ThreadX and NetX, and creates the SNMP Agent and application thread for example, my_snmp_agent_thread_entry().
Time slicing interval	1	Minimum time slice for how long thread runs

Stack size less than 4k may not be large enough. This will be evidenced if the thread entry application makes an SNMP API call for example, nx_snmp_agent_trapv2_send and the system hangs (crashes). It is recommended not to use SNMP Agent Thread because that is the default name that the auto generated code gives the SNMP Agent task thread. The application thread priority should generally be less than the IP thread task for best network performance.

Table 5 Configuration for the NetX and NetX Duo IP Instance

ISDE Property	Setting	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192, 168, 0, 2	SNMP Agent IPv4 Address on the local network where the module guide project will run
Subnet Mask (use commas for separation)	255, 255, 255, 0	Subnet Mask selection
Default Gateway Address (use commas for separation)	0, 0, 0, 0	If the SNMP Manager is not on the local network, this must be a non-zero IP address
IPv6 Global Address** (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	SNMP Agent IPv6 Address (should not be the same as the SNMP Manager which also defaults to the same address!) if using IPv6
IPv6 Link Local Address**(use commas for separation, all zeros mean use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	Link local address is set automatically based on MAC address; used only for IPv6 networks.
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection. This is the recommended setting but it can be modified if required.
IP Helper Thread Priority	3	IP Helper Thread Priority selection. In a busy network environment or an application requiring very high throughput, this priority may be set to 1.
ARP	Enable (Locked)	ARP selection
ARP Cache Size (bytes)	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable (Default: Disable)	Reverse ARP selection
TCP	Enable, Disable (Default: Enable)	TCP selection
UDP	Enable (Locked)	UDP selection
ICMP	Enable, Disable (Default: Enable)	ICMP selection
IGMP	Enable, Disable (Default: Enable)	IGMP selection
IP fragmentation	Enable, Disable (Default: Disable)	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable (Default: Enable)	Auto initialization selection
Link status change callback	NULL	Link status change callback selection

**NetX Duo IP instance only

Table 6 Configuration for the NetX Common

ISDE Property	Setting	Description
No configurable properties		

Table 7 Configuration for NetX Packet Pool Instance on g_packet_pool0

ISDE Property	Setting	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1568	Packet size selection; Set to >= device MTU as these packets will be used to receive packets which may have as much data as the MTU.
Number of Packets in Pool	16	Number of packets in pool selection. 16 packets will suffice but if heavier network traffic is expected, this number can be increased to 32 or higher.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable (Default: Enable)	Auto initialization selection

Table 8 Configuration for NetX Port Ether

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin	IOPORT_PORT_08_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest)-15 (lowest), Disabled (Default: Priority 12)	Ethernet interrupt priority selection; the priority used for this module guide project is Priority 12. This is lower priority than the IP thread task which is set to 1 (recommended)
Link status monitoring method	PHY Polling	Link status monitoring method
Name	g_sf_el_nx	Module name
Channel	1	Channel selection
MAC address change callback	NULL	MAC address change callback selection
Unknown packet receive Callback	NULL	Unknown packet receive callback selection

Note: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

4.2 Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set by using the SSP configurator clock tab, prior to a build, or by using the CGC Interface at run-time.

4.3 Pin Configuration

The ETHERC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the next table illustrates an example selection for the pins.

Note: For some peripherals, the Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Table 9 Pin Selection Sequence for ETHERC1 Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select **Peripherals > Connectivity:ETHERC > ETHERC1.RMII**

Note: The above selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Table 10 Pin Configuration Settings for ETHERC1

Pin Configuration Property	Settings	Description
Operation Mode	Disabled, Custom, RMII (Default: Disabled)	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only (Default: _A only)	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

5. Using the NetX Duo SNMP Agent in an Application

The steps for creating NetX Duo SNMP Agent in the Configurator are:

1. Add NetX Duo SNMP Agent Instance by choosing **X-Ware** -> **NetX Duo** -> **Protocols** -> **NetX Duo SNMP Agent, or X-Ware** -> **NetX** -> **Protocols** -> **NetX SNMP Agent** in the NetX environment. This automatically adds the IP instance and IP default packet pool g_packet_pool0.
2. Click on **Add Network Driver** and choose NetX Port Ether. It should look like the figure below (the MD5, DES and SHA1 block is for the authentication which requires encryption).

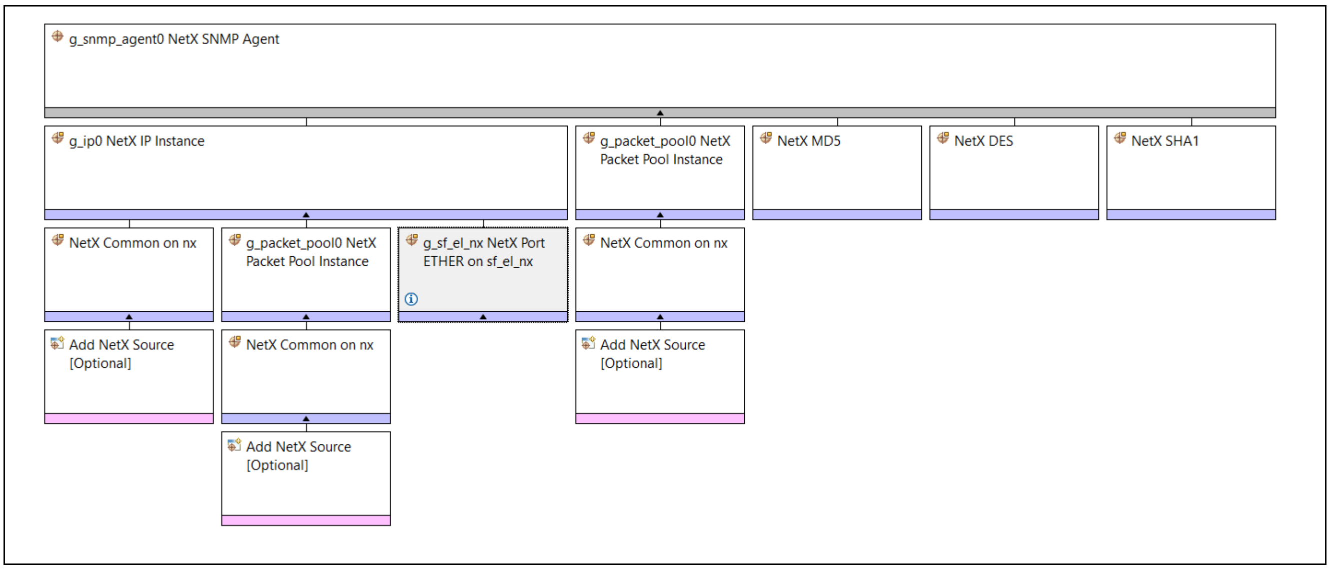

g_snmp_agent0 NetX SNMP Agent
g_ip0 NetX IP Instance
g_packet_pool0 NetX Packet Pool Instance
NetX MD5
NetX DES
NetX SHA1
NetX Common on nx
g_packet_pool0 NetX Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
NetX Common on nx
Add NetX Source [Optional]
NetX Common on nx
Add NetX Source [Optional]
Add NetX Source [Optional]

Figure 5 Thread Stack for the NetX SNMP Agent

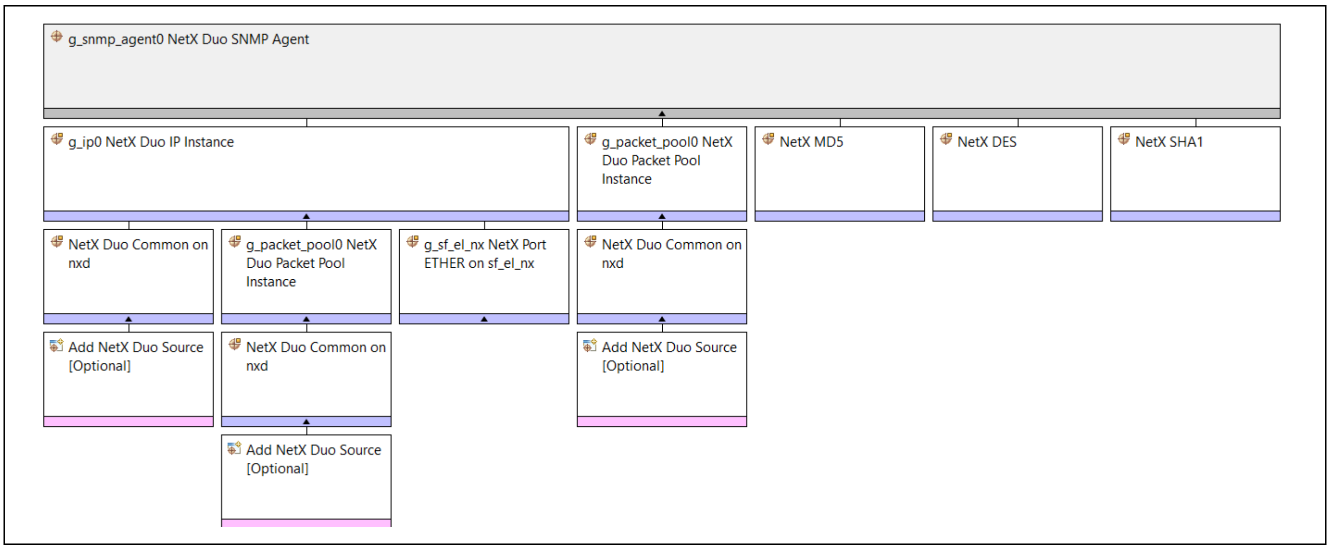

g_snmp_agent0 NetX Duo SNMP Agent
g_ip0 NetX Duo IP Instance
g_packet_pool0 NetX Duo Packet Pool Instance
NetX MD5
NetX DES
NetX SHA1
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
Add NetX Duo Source [Optional]

Figure 6 Thread Stack for the NetX Duo SNMP Agent

3. Wait for the network link to be enabled with the nx_ip_status_check API.
4. If not using SNMPv3 security, skip to step 12 to start the SNMP Agent. If the Auto Initialization property is not set or if the public and private community names are not set in the respective property settings, make sure these steps are performed:
   A. Set the public community name using the nx_snmp_agent_public_string_set API
   B. Set the private community name using the nx_snmp_agent_private_string_set API.
5. Set the SNMP Agent ("engine") context ID using the nx_snmp_agent_context_engine_set API. This is typically a MAC address but it can be any octet string.
6. Set the SNMP Agent boot count in the application source file using the nx_snmp_agent_v3_context_boots_set API.
7. If authentication is desired, create a key for authentication using the nx_snmp_agent_md5_key_create API for an MD5 key, or the nx_snmp_agent_sha_key_create API for an SHA1 key.
8. Register the key to the SNMP Agent using the nx_snmp_agent_authenticate_key_use API.
9. If privacy/encryption is desired, create another key, using the either of the two API mentioned above.
10. Register the key to the SNMP Agent using the nx_snmp_agent_privacy_key_use API.
11. For trap authentication, create a key for authentication using the nx_snmp_agent_md5_key_create API for an MD5 key, or the nx_snmp_agent_sha_key_create API for an SHA1 key and register it with nx_snmp_agent_auth_trap_key_use.

12. For trap privacy, create a key for privacy using the nx_snmp_agent_md5_key_create API for an MD5 key, or the nx_snmp_agent_sha_key_create API for an SHA key and register it with nx_snmp_agent_priv_trap_key_use.
13. Ensure that the SNMP Manager has the same security settings and passwords for all the keys as the SNMP Agent.
14. Start the SNMP Agent by calling the nx_snmp_agent_start API.
15. To send a trap message for SNMPv3 once the SNMP thread task is started, use the nx_snmp_agent_trapv3_send API. The module guide project demonstrates how to create the objects and add them to an SNMPv3 trap message.
16. To send a trap message in SNMPv2, use the nx_snmp_agent_trapv2_send API. The module guide project demonstrates how to create the objects and add them to an SNMPv2 trap message.
17. Stop the SNMP Agent task at any time by calling nx_snmp_agent_stop.

The above common steps are illustrated in a typical operational flow diagram in the following figure.

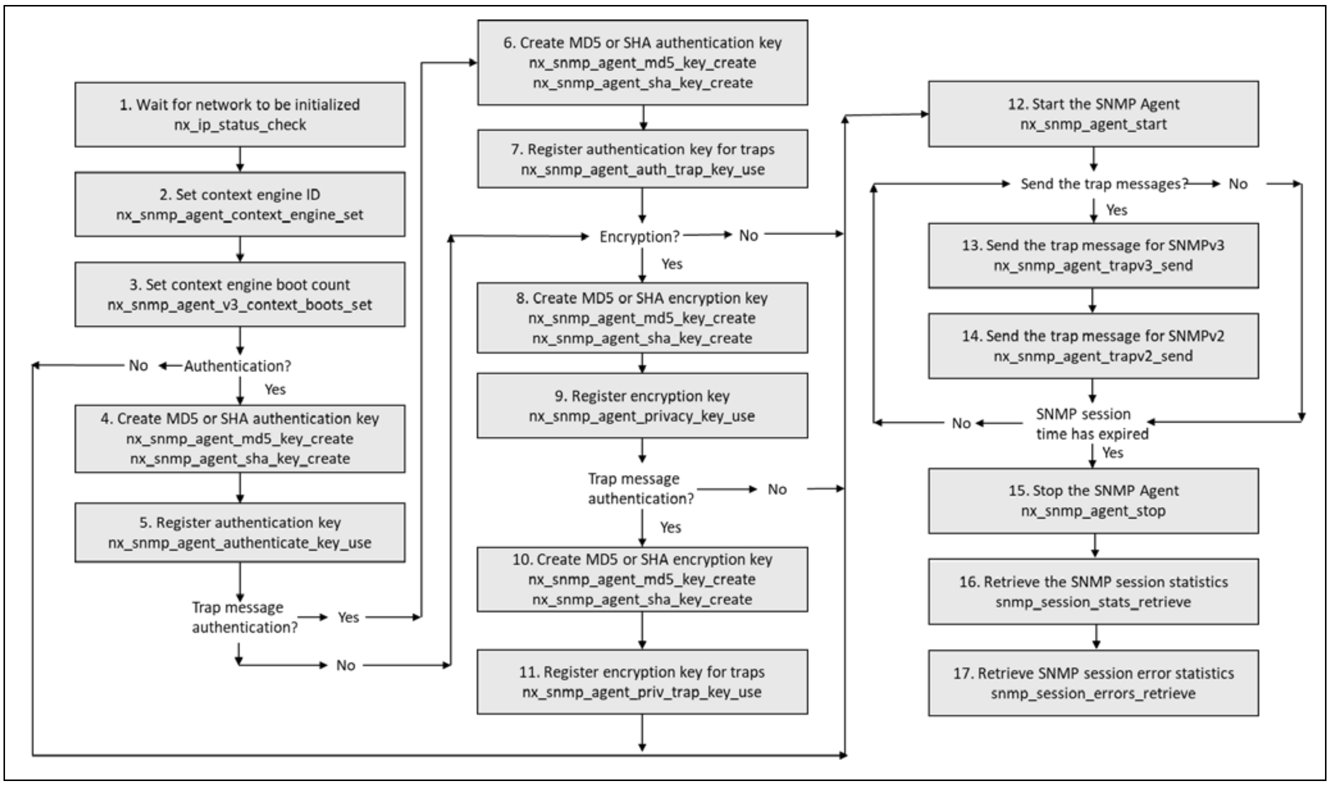

1. Wait for network to be initialized
nx_ip_status_check
2. Set context engine ID
nx_snmp_agent_context_engine_set
3. Set context engine boot count
nx_snmp_agent_v3_context_boots_set
Authentication?
No
Yes
4. Create MD5 or SHA authentication key
nx_snmp_agent_md5_key_create
nx_snmp_agent_sha_key_create
5. Register authentication key
nx_snmp_agent_authenticate_key_use
Trap message authentication?
Yes
No
6. Create MD5 or SHA authentication key
nx_snmp_agent_md5_key_create
nx_snmp_agent_sha_key_create
7. Register authentication key for traps
nx_snmp_agent_auth_trap_key_use
Encryption?
No
Yes
8. Create MD5 or SHA encryption key
nx_snmp_agent_md5_key_create
nx_snmp_agent_sha_key_create
9. Register encryption key
nx_snmp_agent_privacy_key_use
Trap message authentication?
No
Yes
10. Create MD5 or SHA encryption key
nx_snmp_agent_md5_key_create
nx_snmp_agent_sha_key_create
11. Register encryption key for traps
nx_snmp_agent_priv_trap_key_use
12. Start the SNMP Agent
nx_snmp_agent_start
Send the trap messages?
No
Yes
13. Send the trap message for SNMPv3
nx_snmp_agent_trapv3_send
14. Send the trap message for SNMPv2
nx_snmp_agent_trapv2_send
SNMP session time has expired
No
Yes
15. Stop the SNMP Agent
nx_snmp_agent_stop
16. Retrieve the SNMP session statistics
snmp_session_stats_retrieve
17. Retrieve SNMP session error statistics
snmp_session_errors_retrieve

Figure 7 Flow Diagram of a Typical NetX Duo SNMP Agent Application

6. NetX Duo SNMP Agent Application Project

The Application Project associated with this Module Guide demonstrates the above steps in a full design. The project can be found using the link provided in the Reference Section at the end of this document. You may want to import and open the Application Project within ISDE and view the configuration settings for the NetX Duo SNMP Agent Module. You can also read over the code in snmp_agent_thread_entry.c and NetXDuo_SNMP_Agent_mg_ap.c*, which is used to illustrate the NetX Duo SNMP Agent APIs in a complete design.

*While the file name specifies NetX Duo, the same file will work fine in a NetX environment.

The Application Project demonstrates the typical use of the NetX Duo SNMP Agent APIs. The Application Project main thread entry initializes the NetX Duo SNMP Agent Framework. After waiting for the network to be enabled, it then sets up and runs an SNMP session for a specified time period. After the time expires, the SNMP Agent thread task is stopped, and the application reports the session statistics as well as any errors that occurred.

Table 11 Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	6.2.0	Integrated Solution Development Environment
SSP	1.4.0	Synergy Software Platform
IAR EW for Synergy	8.21.1	IAR Embedded Workbench® for Renesas Synergy™
SSC	6.2.0.R20180102	Synergy Standalone Configurator

Resource	Revision	Description
SK-S7G2	v3.0 to v3.3	Starter Kit

The snmp_agent_thread_entry.c, NetXDuo_SNMP_Agent_mg_ap.c, and NetXDuo_SNMP_Agent_mg_ap.h files are located in the project once it has been imported into the ISDE. You can open these files within the ISDE and follow along with the description provided to help identify key uses of APIs.

The properties for the SNMP Agent and IP stack are described in section 4. The entries which are highlighted in yellow may need to be changed to suit your application environment.

The flow diagram of the SNMP Agent module guide application project is shown in the following figure.

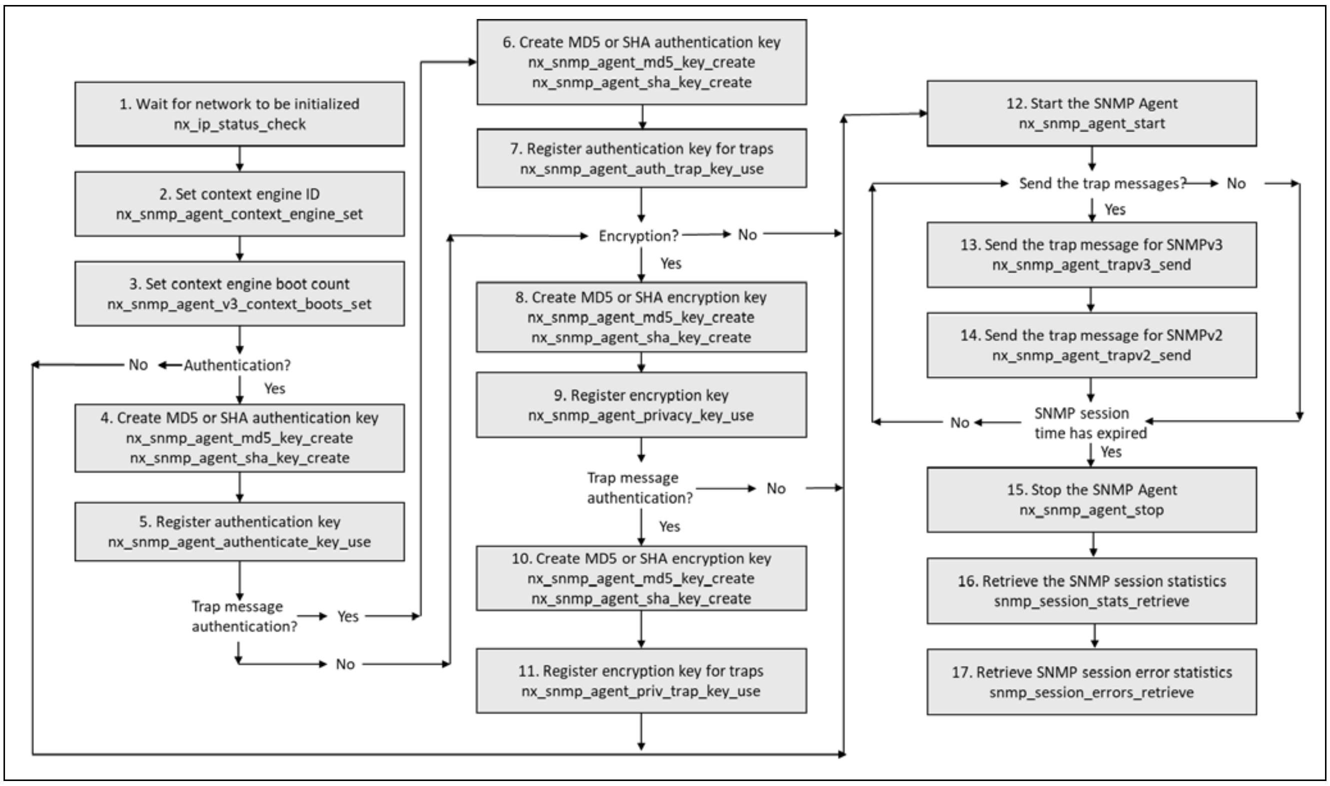

1. Wait for network to be initialized nx_ip_status_check
2. Set context engine ID nx_snmp_agent_context_engine_set
3. Set context engine boot count nx_snmp_agent_v3_context_boots_set
Authentication?
No
Yes
4. Create MD5 or SHA authentication key nx_snmp_agent_md5_key_create nx_snmp_agent_sha_key_create
5. Register authentication key nx_snmp_agent_authenticate_key_use
Trap message authentication?
Yes
No
6. Create MD5 or SHA authentication key nx_snmp_agent_md5_key_create nx_snmp_agent_sha_key_create
7. Register authentication key for traps nx_snmp_agent_auth_trap_key_use
Encryption?
No
Yes
8. Create MD5 or SHA encryption key nx_snmp_agent_md5_key_create nx_snmp_agent_sha_key_create
9. Register encryption key nx_snmp_agent_privacy_key_use
Trap message authentication?
No
Yes
10. Create MD5 or SHA encryption key nx_snmp_agent_md5_key_create nx_snmp_agent_sha_key_create
11. Register encryption key for traps nx_snmp_agent_priv_trap_key_use
12. Start the SNMP Agent nx_snmp_agent_start
Send the trap messages?
No
Yes
13. Send the trap message for SNMPv3 nx_snmp_agent_trapv3_send
14. Send the trap message for SNMPv2 nx_snmp_agent_trapv2_send
SNMP session time has expired
No
Yes
15. Stop the SNMP Agent nx_snmp_agent_stop
16. Retrieve the SNMP session statistics snmp_session_stats_retrieve
17. Retrieve SNMP session error statistics snmp_session_errors_retrieve

Figure 8 NetX Duo SNMP Agent Application Project Flow Diagram

In the module guide project, the SNMP Agent is enabled for SNMPv2 and SNMPv3, which means that it will respond to both SNMPv2 and SNMPv3 queries. It is also enabled for authentication and encryption. Lastly, it is also set up to send a series of trap messages (in SNMPv2 and SNMPv3) to the SNMP Manager.

The snmp_agent_thread_entry function waits for the network to be enabled (#1 of the flow chart). The SNMP Agent is already created but not started. For the SNMPv3 protocol, however, it must also set the context engine ID and number of boots before starting (#2 and #3).

It then sends a set of parameters in the run_snmp_session function for setting up and running an SNMP session as follows:

Create and register an MD5 authentication key (#4 and #5) for SNMP manager queries using the following APIs:

- `nx_snmp_agent_md5_key_create`
- `nx_snmp_agent_authenticate_key_use`

Create and register an MD5 authentication key for SNMPv3 trap messages (#6 and #7) for SNMP manager queries using the following APIs:

- `nx_snmp_agent_md5_key_create`
- `nx_snmp_agent_auth_trap_key_use`

Create and register an MD5 encryption key for SNMP Manager queries (#8 and #9) for SNMP manager queries using the following APIs:

- `nx_snmp_agent_md5_key_create`
- `nx_snmp_agent_privacy_key_use`

Create and register an MD5 encryption key for SNMPv3 trap messages (#10 and #11) for SNMP manager queries using the following APIs:

- `nx_snmp_agent_md5_key_create`
- `nx_snmp_agent_priv_trap_key_use`

Authentication and encryption need not use the same algorithm but it is recommended practice to do so. The SNMP Agent thread task can now be started (step 12) by calling the nx_snmp_agent_start API.

The input to run_snmp_session also specifies if the SNMP Agent should send traps and how long to let the SNMP thread task run (in timer ticks) before stopping the SNMP session. If send_traps is set to NX_TRUE, the following two APIs are used to send SNMPv2 and SNMPv3 traps (step 13 and step 14):

- `nx_snmp_agent_trapv3_send`
- `nx_snmp_agent_trapv2_send`

When the session time expires, run_session_snmp stops the session by calling nx_snmp_agent_stop (step 15) AND returns to the caller.

Then, run_snmp_session calls a utility function, snmp_session_stats_retrieve (step 16), to report session statistics, such as the number of requests received, responses sent, and traps sent. It also calls a function, snmp_session_errors_retrieve (step 17) to check for SNMP session errors (bad packets received, internal errors, and so forth) tracked by the SNMP Agent handling SNMP message processing. These errors are different from the errors, if there are any, that run_snmp_session reports back to snmp_agent_thread_entry as a result of creating session keys and sending traps.

The debug output uses the printf utility in the Renesas Virtual Debug console in Synergy or the Terminal IO option in IAR.

Note: The above description assumes that you are familiar with using printf() the Debug Console in the Synergy Software Package. If you are unfamiliar with this, refer to the “How do I Use Printf() with the Debug Console in the Synergy Software Package Knowledge Base article, as described in the References section at the end of this document. Alternatively, the user can see results using the watch variables in the debug mode.

7. Customizing NetX Duo SNMP Agent for a Target Application

The NetX Duo SNMP Agent project shares a packet pool with the IP instance. That packet pool has only 16 packets for both IP thread tasks, receiving packets and SNMP Agent sending messages to the server. In a real-world application, you might need to increase the number of packets particularly if the SNMP manager(s) make GET BULK queries which ‘walk’ the MIB database tree and generate a large flow of packets.

Encryption may not be necessary, as authentication at least guarantees that the data is not tampered with. Encryption requires considerable processing and can slow down throughput, so if it is not needed, it is recommended to disable it.

8. Running the NetX Duo SNMP Agent Application Project

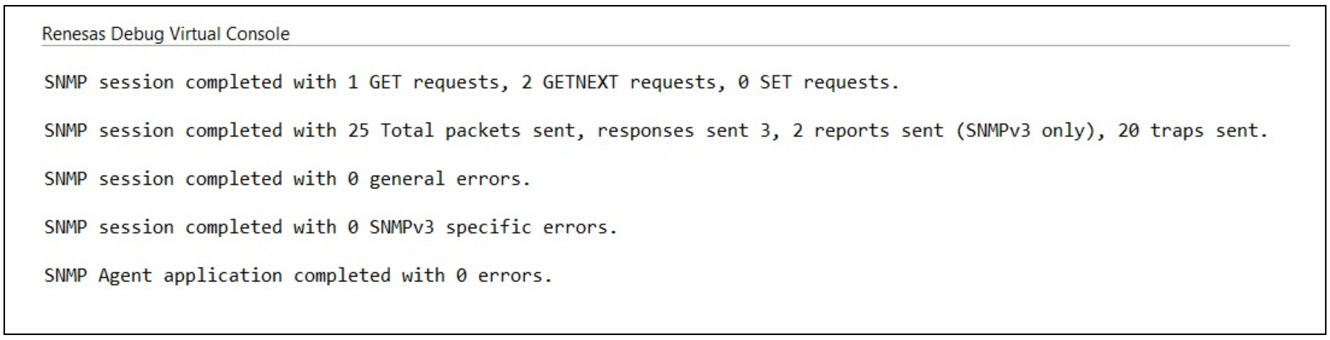
To run the NetX Duo SNMP Agent Application project and to see it executing on a target kit, you can simply import it into your ISDE, compile and run debug.

Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few chapters of the *SSP User’s Manual* for a description on how to accomplish these steps.

1. Import and build the example project included with this module guide according to the Synergy Project Import Guide (11an0023eu0120-synergy-ssp-import-guide.pdf).
2. Connect to the host PC via a micro USB cable to J19 on SK-S7G2.
3. Connect an ethernet cable to J11 to the local network (if using Ethernet).
4. Start the application.
5. Start the SNMP Manager (or at least have it initiate contact with the SNMP Agent).
6. For up to the length of time specified by the run_snmp_session function, send queries to the SNMP Agent from the SNMP Manager.
7. Verify that the SNMP Manager makes a successful “Discovery” (establishes a connection) with the SNMP Agent and accepts trap messages. Most SNMP browser (manager) tools have a log utility to track messaging between the Agent and Manager.

Note: It is beyond the scope of this document to discuss how to use an SNMP Manager. It is assumed that the user is already familiar with SNMP protocol and SNMP tools. The SNMP Manager (browser) used for this project is the MG-Soft MIB Browser Professional Developer's Edition. Be aware that cheap or free SNMP utilities sometimes do not always properly follow SNMPv3 protocol.

8. When run_snmp_session and snmp_agent_thread_entry have completed, the output can be viewed in the Renesas Debug Console. A successful session from the SNMP project run in the e^2 studio environment is shown in Figure 9.

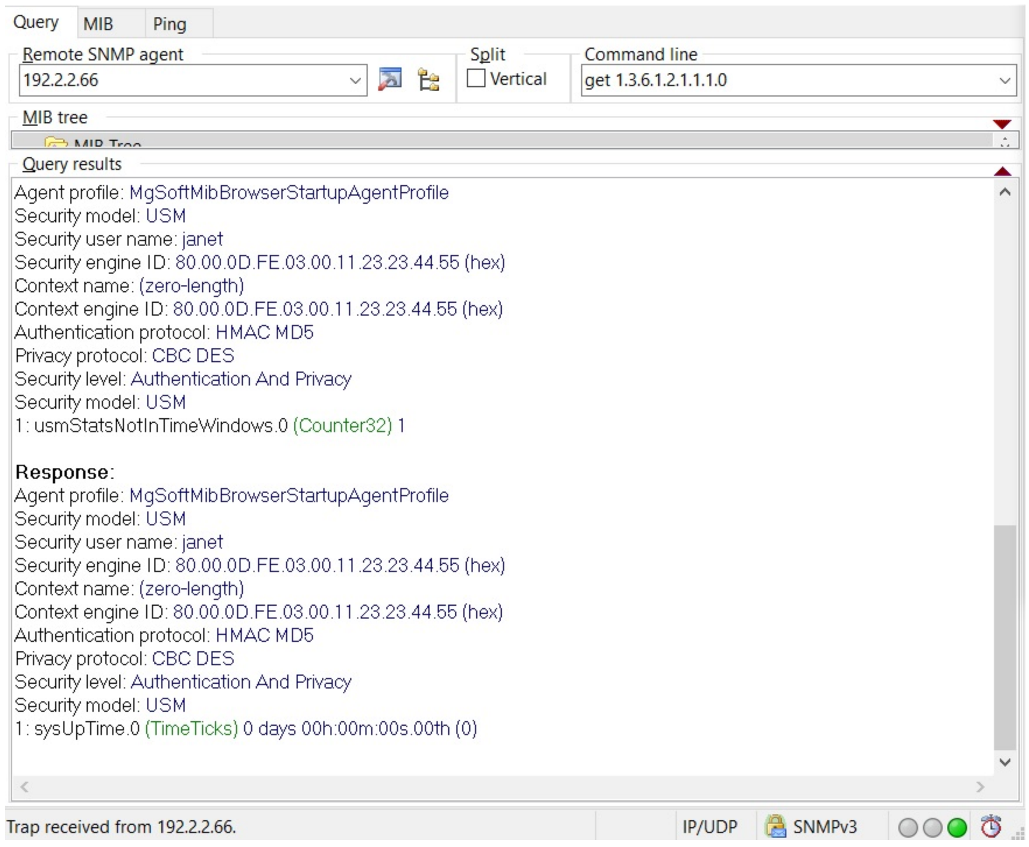


Renesas Debug Virtual Console

```
SNMP session completed with 1 GET requests, 2 GETNEXT requests, 0 SET requests.

SNMP session completed with 25 Total packets sent, responses sent 3, 2 reports sent (SNMPv3 only), 20 traps sent.

SNMP session completed with 0 general errors.

SNMP session completed with 0 SNMPv3 specific errors.

SNMP Agent application completed with 0 errors.
```

Figure 9 Debug Console Output from a Successful NetX Duo SNMP Agent Session

Figure 10 shows the log capture of SNMP Manager 'handshake' with NetX Duo SNMP Agent Project from the MG-Soft MIB Browser application. The green light on the lower right indicates that the Discovery process was successful. The security level indicates both Authentication and Privacy (encryption).

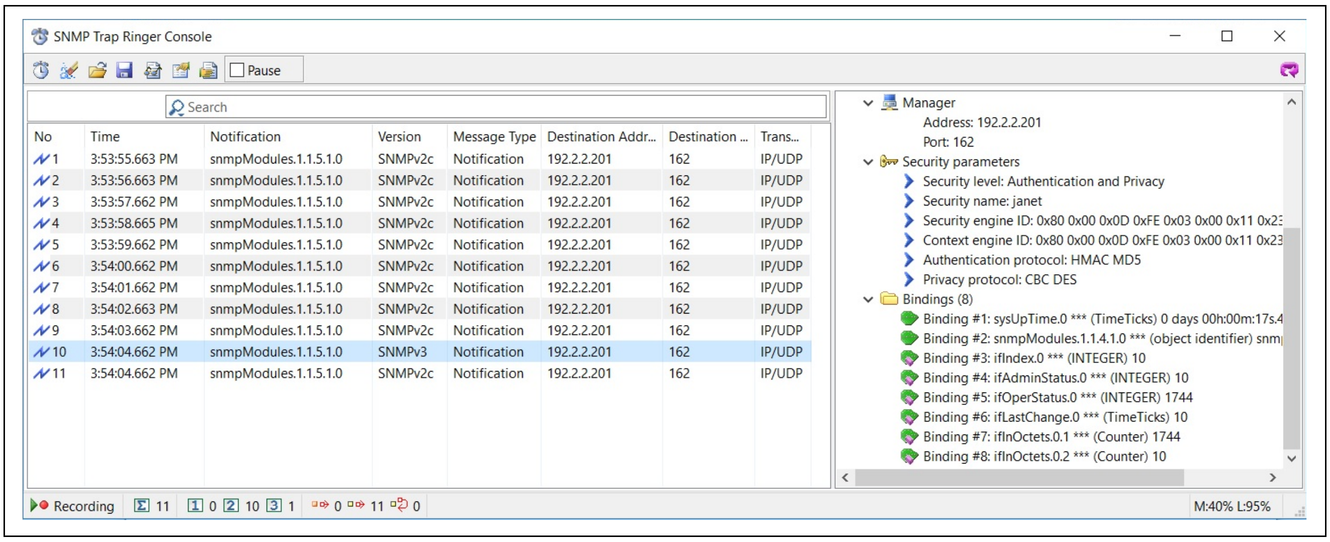


```
Agent profile: MgSoftMibBrowserStartupAgentProfile
Security model: USM
Security user name: janet
Security engine ID: 80.00.0D.FE.03.00.11.23.23.44.55 (hex)
Context name: (zero-length)
Context engine ID: 80.00.0D.FE.03.00.11.23.23.44.55 (hex)
Authentication protocol: HMAC MD5
Privacy protocol: CBC DES
Security level: Authentication And Privacy
Security model: USM
1: usmStatsNotInTimeWindows.0 (Counter32) 1

Response:
Agent profile: MgSoftMibBrowserStartupAgentProfile
Security model: USM
Security user name: janet
Security engine ID: 80.00.0D.FE.03.00.11.23.23.44.55 (hex)
Context name: (zero-length)
Context engine ID: 80.00.0D.FE.03.00.11.23.23.44.55 (hex)
Authentication protocol: HMAC MD5
Privacy protocol: CBC DES
Security level: Authentication And Privacy
Security model: USM
1: sysUpTime.0 (TimeTicks) 0 days 00h:00m:00s.00th (0)
```

Trap received from 192.2.2.66. IP/UDP SNMPv3

Figure 10 Log capture of SNMP Manager 'handshake' with NetX Duo SNMP Agent Project from the MG-Soft MIB Browser application

Figure 11 shows the log capture of SNMP Manager capture from the MG-Soft MIB Browser of the SNMP v2 trap and the SNMP v3 trap messages from the NetX Duo SNMP Agent Project from the MG-Soft MIB Browser application. The highlighted entry of an SNMP v3 trap message shows both Authentication and Privacy (encryption).


SNMP Trap Ringer Console

Pause

Search

No	Time	Notification	Version	Message Type	Destination Addr...	Destination ...	Trans...
1	3:53:55.663 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
2	3:53:56.663 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
3	3:53:57.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
4	3:53:58.665 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
5	3:53:59.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
6	3:54:00.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
7	3:54:01.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
8	3:54:02.663 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
9	3:54:03.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP
10	3:54:04.662 PM	snmpModules.1.1.5.1.0	SNMPv3	Notification	192.2.2.201	162	IP/UDP
11	3:54:04.662 PM	snmpModules.1.1.5.1.0	SNMPv2c	Notification	192.2.2.201	162	IP/UDP

```
Manager
    Address: 192.2.2.201
    Port: 162
Security parameters
    Security level: Authentication and Privacy
    Security name: janet
    Security engine ID: 0x80 0x00 0x0D 0xFE 0x03 0x00 0x11 0x23
    Context engine ID: 0x80 0x00 0x0D 0xFE 0x03 0x00 0x11 0x23
    Authentication protocol: HMAC MD5
    Privacy protocol: CBC DES
Bindings (8)
    Binding #1: sysUpTime.0 *** (TimeTicks) 0 days 00h:00m:17s.4
    Binding #2: snmpModules.1.1.4.1.0 *** (object identifier) snm
    Binding #3: ifIndex.0 *** (INTEGER) 10
    Binding #4: ifAdminStatus.0 *** (INTEGER) 10
    Binding #5: ifOperStatus.0 *** (INTEGER) 1744
    Binding #6: ifLastChange.0 *** (TimeTicks) 10
    Binding #7: ifInOctets.0.1 *** (Counter) 1744
    Binding #8: ifInOctets.0.2 *** (Counter) 10
```

Recording Σ 11 1 0 2 10 3 1 0 11 0 M:40% L:95%

Figure 11 Log capture of SNMP Manager capture from MG-Soft MIB Browser of SNMPv2 trap and SNMPv3 trap messages from the NetX Duo SNMP Agent Project from the MG-Soft MIB Browser application

9. Conclusion

This Module Guide has provided all the background information needed to select, add, configure and use the module in an example project. Many of these steps were time consuming and error prone activities in previous generations of embedded systems. The Renesas Synergy Platform makes these steps much less time consuming and removes the common errors, like conflicting configuration settings or incorrect selection of low level drivers. The use of high level APIs, as demonstrated in the Application Project illustrate additional development time savings by allowing work to begin at a high level, avoiding the time required in older development environments to use or, in some cases, create low level drivers.

10. Next Steps

After you have mastered this NetX Duo SNMP Agent project, you will need to acquire or build your own MIB databases and add logic for updating the information in the database in real time. The GET, GETNEXT, SET handlers may require greater complexity to handle real world demands rather than simply report back a fixed value as they do in this project. There are hundreds of standardized MIB databases for all types of industry networks. Trap messages should be generated from real time events from the outside world rather than the simple traps sent in the project.

11. Reference Information

To find the most up to date reference materials and their locations, visit the Synergy Knowledge Base and do a search for the module name and include “module guide references” in the search.

For example, if you are looking for the References for the r_doc module, visit https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base and enter “r_doc module guide references” in the search bar. The search will bring up a list of results, and the top one will be the References Page for that Module Guide. The following URL will take you directly to the search results for the example.

https://en-us.knowledgebase.renesas.com/Special:Search?fpid=230&search=r_doc%20module%20guide&path=&limit=55&page=1&q=r_doc%20module%20guide%20references&tags=

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	renesassynergy.com/software
Synergy Software Package	renesassynergy.com/ssp
Software add-ons	renesassynergy.com/addons
Software glossary	renesassynergy.com/softwareglossary
Development tools	renesassynergy.com/tools
Synergy Hardware	renesassynergy.com/hardware
Microcontrollers	renesassynergy.com/mcus
MCU glossary	renesassynergy.com/mcuglossary
Parametric search	renesassynergy.com/parametric
Kits	renesassynergy.com/kits
Synergy Solutions Gallery	renesassynergy.com/solutionsgallery
Partner projects	renesassynergy.com/partnerprojects
Application projects	renesassynergy.com/applicationprojects
Self-service support resources:	
Documentation	renesassynergy.com/docs
Knowledgebase	renesassynergy.com/knowledgebase
Forums	renesassynergy.com/forum
Training	renesassynergy.com/training
Videos	renesassynergy.com/videos
Chat and web ticket	renesassynergy.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct 24, 2018	-	Initial version

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.2