SnmpLite

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

RTAC LIBRARY

SnmpLite

Introduction

This library encapsulates common communications management responsibilities that industrial control and monitoring equipment are expected to perform on the communications fabric of a control system using Simplified Network Management Protocol (SNMP).

The common use for these devices is to annunciate failures within the system. Therefore, this library only provides basic network interface descriptions and port states to allow detection and annunciation of device failures. More advanced diagnostic and system description tools that fully use SNMP are outside the scope of this library, and are not considered to be common requirements of an automation controller.

Glossary

Abstract Syntax Notation One (ASN.1) A standard that provides rules and a notation for representing, encoding, transmitting, and decoding data in telecommunications. SNMP uses this standard to represent the data within a UDP packet.

Management Information Base (MIB) A hierarchical description of the structure and data available in a device that supports SNMP.

Object Identifier (OID) A unique identifier for an object or leaf in a hierarchically defined namespace, where a "." specifies the next object as the child of the previous. For example, the OID 1.3.6.1.2.1.2.2.1 identifies the entries in a table of interfaces. In the context of this library, OIDs uniquely identify information found in communication with the network devices.

SNMP Trap Unsolicited information SNMP agents send to SNMP managers. Events that trigger SNMP Traps and the Trap content are configured on the SNMP agent device and utilize Port 162.

Simplified Network Management Protocol (SNMP) An Internet standard protocol for managing devices on IP networks. This protocol is generally supported by network-fabric devices such as Ethernet switches and routers, and it is often used by other network-centric devices. There are many versions of SNMP. All SNMP traffic, except SNMP traps, is sent and received on Port 161.

Simple Network Management Protocol Version 2, Community-Based SNMPv2c) This version of the protocol transmits all data, including the community strings used to group SNMP agents and managers, in raw form with no encryption.

User Datagram Protocol (UDP) A connectionless, packet-based protocol used to stream data between two devices without requiring responses. UDP does not implement handshaking, packet ordering, or confirmation of packet delivery.

Special Considerations

Consider the following topics when implementing this library in a project.

Versions of SNMP

This version of the library only supports SNMPv2c, which is an unsecured implementation that transmits all data on the wire without encryption or security.

Open SNMP Ports

This library does not have the ability to open ports in the RTAC firewall to communicate with the various devices. When the class_SnmpManager is instantiated in an ACSELERATOR RTAC project, the creator of that project is responsible for creating two access points that allow the class_SnmpManager object to communicate with the specified devices.

- ➤ UDP Port 161 Ethernet Listener Access Point must be created for all normal SNMP traffic.
- ➤ UDP Port 162 Ethernet Listener Access Point must be created to receive SNMP traps.

Declaring Objects in Global Variable Lists

This library uses variables in a Global Variable List (GVL) set to Global Initialization Slot 1 as the registry mechanism for its classes.

Instantiating a class in this library as a global variable and forcing it to be initialized at Global Initialization Slot 1 or lower will cause undefined behavior. Declaring classes in a normal global variable list causes no issues because the default Global Initialization Slot is 50000 (unless explicitly set otherwise).

All SNMP Objects in a Single, Non-Time Critical Task

This library expects that all instantiated objects are in the same task. Care has been taken in designing this library to evenly distribute processing load, and the library scales to management of many devices without incurring additional processing. However, the execution time to parse arbitrarily large and complex responses from agent devices is, by nature, not deterministic. Therefore, this library should not be used on the same task with logic that must execute deterministically.

Put all SNMP Objects in a Single Program

Consider instantiating all of these objects in a single IEC 61131 program to mitigate the concerns described in *Declaring Objects in Global Variable Lists on page 4* and *All SNMP Objects in a Single, Non-Time Critical Task on page 4*.

Copying Instantiated Objects

Copying classes from this library causes unwanted behavior. This means the following:

1. The assignment operator ":=" must not be used on any class from this library; consider assigning pointers to the objects instead.

```
// This is bad and in most cases will provide a compiler error such
    as:
// "C0328: Assignment not allowed for type class_SnmpObject"
mySnmpObject := otherSnmpObject;
```

```
// This is fine
someVariable := mySnmpObject.value;
// As is this
pt_mySnmpObject := ADR(mySnmpObject);
```

2. Classes from this library must never be VAR_INPUT or VAR_OUTPUT members in function blocks, functions, or methods. Place them in the VAR_IN_OUT section or use pointers instead.

Scope of Instantiated Objects

Classes in this library have memory allocated inside them. As such, they should only be created in environments of permanent scope (e.g., Programs, Global Variable Lists, or VAR_STAT sections).

Supported Firmware Versions

You can use this library on any device configured using ACSELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.0.3 and older can be used on RTAC firmware version R133 and higher.

Enumerations

Enumerations make code more readable by allowing a specific number to have a readable textual equivalent.

NOTE: See http://tools.ietf.org/html/rfc2863 for more information about the interface states.

enum_InterfaceStatus

This enumeration provides a textual representation of the statuses possible for any individual port.

Enumeration	Description
NONE	Interface specified is out of range for this device.
UP	Interface status is up.
DOWN	Interface status is down.
TESTING	Interface is in test configuration.
UNKNOWN	Port link status is unknown.
DORMANT	The interface is ready to transmit and receive network traffic, but is waiting on some external event.
NOT_PRESENT	Interface is not configured in this hardware.
LOWER_LAYER_DOWN	The dependent interface is not up, a condition relevant only if stacked interfaces are employed.

Structures

Structures provide a means to group together several memory locations (variables), making them easier to manage.

struct_InterfaceInfo

NOTE: iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) ifTable(2) . ifEntry(1) . item(N)

This structure contains some of the commonly used information available in the table OID 1.3.6.1.2.1.2.2.1.N.

Name	IEC 61131 Type	Description
ifIndex	UDINT	The index the manufacturer provides from OID 1.3.6.1.2.1.2.2.1.1.N.
ifDescr	STRING(255)	First 255 characters of description from OID 1.3.6.1.2.1.2.2.1.2.N.
ifType	UDINT	The integer value of the interface type from OID 1.3.6.1.2.1.2.2.1.3.N.
ifSpeed	UDINT	The bandwidth this interface report during initialization from OID 1.3.6.1.2.1.2.2.1.5.N.
ifPhysicalAddress	STRING(80)	Usually the MAC address. Obtained from OID 1.3.6.1.2.1.2.2.1.6.N.
ifOperStatus	enum_InterfaceStatus	The operational status of this link from OID 1.3.6.1.2.1.2.2.1.8.N. This updates periodically, or when this library receives a trap.

Classes

class_SnmpManager (Class)

A single class_SnmpManager object is instantiated to manage the UDP traffic on Port 161 for SNMP get and set requests and responses to these requests. The same object listens on Port 162, which is the SNMP trap listening port. If two or more class are instantiated, all but one of the classes will fail to initialize and display an appropriate error state.

This object issues SNMP requests queued by instantiated SNMP agent classes. It sends responses to these requests and forwards SNMP traps received from a given IP address to the SNMP agent class associated with the sending IP. The agent class is responsible for the processing of the message.

The manager issues no more than one SNMP request per call to the `Run()` method. This constraint evenly distributes the load per scan and allows support for an arbitrary number of agents. However, this constraint also means that the more devices that this manager supervises, the more time it takes to cycle through the pending requests from all of the agent classes. This result can be mitigated by configuring traps on the SNMP agent devices because the manager also processes one trap per call to the `Run()` method; a trap event will update the status of the SNMP device immediately and the delay in getting new information is not affected by the number of devices being monitored. Instead, only the number of traps received simultaneously affects the latency of updates to the agent monitoring classes.

Run (Method)

Call this method once every scan on a low-priority task where real-time performance is not critical.

Processing

The `Run()` method does the following:

- ➤ Issues a pending poll:
  - ➢ Get the next registered agent class.
  - ➢ Issue the queued request.
- ➤ Dequeues a maximum of two SNMP traps from the Port 162 socket and then sends the traps to the agent class designated to the IP address of the sender for parsing.
- ➤ Dequeues a maximum of two SNMP responses from the Port 161 socket and then send the responses to the agent class designated to the IP address of the sender for parsing.

class_SnmpAgentMonitor (Class)

When instantiated, this class registers itself with the active class_SnmpManager. There is no need to call methods on this class or to run it periodically because the manager class is responsible for refreshing the state of all agent monitors.

This class monitors a generic SNMP agent device that has a list of network interfaces listed under OID 1.3.6.1.2.1.2.2.1 [1]. It also displays the number of interfaces the SNMP agent provides in response to requesting the Integer32 value at OID 1.3.6.1.2.1.2.1 [2].

The list of interfaces is initially obtained by walking through all of the table objects. If the number of interfaces returned by walking the interface table is less than the number of interfaces OID 1.3.6.1.2.1.2.1 advertises, the device will not enable. If the interface information from all interfaces can be obtained, then the object enables itself. Subsequently, the agent only requests port statuses via OID 1.3.6.1.2.1.2.2.1.8.N [3].

The port information obtained by `GetInterfaceInfo()` updates only during initialization, with the exception of *ifOperStatus*, which continually updates.

In the event that (a) the agent does not receive replies to requests for *ifOperStatus*, (b) the quantity of interfaces changes, or (c) the indexing of the monitored interfaces changes, this block sets *ENO* to FALSE and attempts to reinitialize.

This basic SNMP agent monitor discards SNMP traps it receives from the class_SnmpManager object because the contents of SNMP traps are highly dependent on the MIB of the particular device sending the traps.

Initialization Inputs

Name	IEC 61131 Type	Description
ipAddress	STRING(15)	The IP address of the monitored agent as a string.
communityString	STRING(80)	A string that designates the community this agent device has been assigned. This ID is included in every packet the agent and manager send.

Outputs

Name	IEC 61131 Type	Description
IpAddress	STRING(15)	The IP address as a string.
ENO	BOOL	The class was initialized.
NumInterfaces	UDINT	The number of interfaces the agent device has.
MessagesReceived	UDINT	The total number of responses and traps received from the manager object since *ENO* last became true.
TimeSinceLastMessage	TIME	The elapsed time since the last message from this device.
Error	STRING(255)	The last error encountered will be described here. It will not be cleared.

GetInterfaceStatus (Method)

This method provides the status of the specified interface.

[1] iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1).
[2] iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifNumber(1).
[3] iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1) . ifOperStatus(8) . item(N).

Inputs

Name	IEC 61131 Type	Description
interfaceIndex	UDINT	A numeric identifier of the interface. It must be greater than 0 and less than or equal to *NumInterfaces*.

Return Value

IEC 61131 Type	Description
enum_InterfaceStatus	The state of the requested interface.

Processing

➤ Check that *interfaceIndex* is within the specified range. Return NONE if it is not.

➤ Return the state of the specified interface.

GetInterfaceInfo (Method)

This method returns the information obtained from walking OID 1.3.6.1.2.1.2.2.1 before the *ENO* output was true.

NOTE: iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1)

Inputs

Name	IEC 61131 Type	Description
interfaceIndex	UDINT	A numeric identifier of the interface. It must be greater than 0 and less than or equal to *NumInterfaces*.

Outputs

Name	IEC 61131 Type	Description
info	struct_InterfaceInfo	The structure containing the interface information.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the object is initialized and the *interfaceIndex* is in range.

Processing

➤ Check that *interfaceIndex* is within the specified range. Return FALSE if it is not, leaving the output in an undefined state.

➤ Copy internal information to the output and return true.

Reinitialize (Method)

Returns the agent to an uninitialized state, resulting in the *ENO* output being set to false. From this point, the agent automatically tries to re-initialize, refreshing all port information in the process.

class_SEL2730MAgent (Class)

This class has all of the functionality of the class_SnmpAgentMonitor. However, because the number of ports are known in advance, the class also displays the port statuses as pins.

Because this class contains knowledge of the device, future changes to SEL-2730M firmware may change interface enumerations in SNMP. All versions of SEL-2730M firmware that do not impact the assumptions in *SEL-2730M Assumptions on page 10* will work.

SEL-2730M Assumptions

1. Port F up/down status is not accessible via SNMP get request. Where all other ports default to NONE and then report the status of SNMP get requests, this port defaults to UNKNOWN until a trap is received and then changes to UP or DOWN as appropriate.
2. Ports 1–24 can be accessed through use of a GetNextRequest SNMP command. The order of the ports is unimportant.
3. The description field of Ports 1–24 always ends with the port number. The OID order is randomly defined at every boot, so the class uses the description to associate the OID with the physical port index.

When instantiated, this class registers itself with the active class_SnmpManager. There is no need to call methods on this class or to run it periodically because the manager class is responsible for refreshing the state of all agent monitors.

This class monitors a generic SNMP agent device that has a list of network interfaces listed under OID 1.3.6.1.2.1.2.2.1 [4]. It also displays the number of interfaces the SNMP agent provides in response to requesting the Integer32 value at OID 1.3.6.1.2.1.2.1 [5].

The list of interfaces is initially obtained by walking through all of the table objects. If the number of interfaces returned by walking the interface table is less than the number of interfaces OID 1.3.6.1.2.1.2.1 advertises, the device will not enable. If the interface information from all interfaces can be obtained, then the object enables itself. Subsequently, the agent only requests port statuses via OID 1.3.6.1.2.1.2.2.1.8.N [6].

The port information obtained by `GetInterfaceInfo()` updates only during initialization, with the exception of *ifOperStatus*, which continually updates.

In the event that (a) the agent does not receive replies to requests for *ifOperStatus*, (b) the quantity of interfaces changes, or (c) the indexing of the monitored interfaces changes, this block sets *ENO* to FALSE and attempts to reinitialize.

[4]iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1).
[5]iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifNumber(1).
[6]iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1) . ifOperStatus(8) . item(N).

Traps from SEL-2730M devices have a known structure, so traps sent to this agent monitor from the class_SnmpManager are parsed. The textual message contained in a trap that is displayed in the *LastMessage* output. If the trap indicates a port status change, the state of the indicated port updates accordingly.

Whether by traps or periodic queries, the output pin statuses update automatically without the implementer having to call any methods or the body of the class.

Initialization Inputs

Name	IEC 61131 Type	Description
ipAddress	STRING(15)	The IP address of the monitored agent as a string.
communityString	STRING(80)	A string that designates the community this agent device has been assigned. This ID is included in every packet the agent and manager send.

Outputs

Name	IEC 61131 Type	Description
IpAddress	STRING(15)	The IP address as a string.
ENO	BOOL	The class was initialized.
NumInterfaces	UDINT	The number of interfaces the agent device has.
MessagesReceived	UDINT	The total number of responses and traps received from the manager object since *ENO* last became true.
TimeSinceLastMessage	TIME	The elapsed time since the last message from this device.
Error	STRING(255)	The last error encountered will be described here. It will not be cleared.
LastMessage	STRING(255)	The first 255 characters of the last string message from this device.
PortStatus	ARRAY[0..24] OF enum_InterfaceStatus	The state of all ports. Index 0 is ETHF, Index 1 is Port 1, Index 24 is Port 24.

GetPortInfo (Method)

This method returns the information obtained from walking OID 1.3.6.1.2.1.2.2.1 before the *ENO* output was true. The index provided is the port number for the SEL-2730M instead of an arbitrary order.

NOTE: iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1).

Inputs

Name	IEC 61131 Type	Description
portNumber	UDINT(0..24)	The port index on the switch. portNumber 0 is for ETHF, 1 is for Port 1, and 24 is for Port 24.

Outputs

Name	IEC 61131 Type	Description
info	struct_InterfaceInfo	The structure containing the interface information.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the object is initialized and the requested port number is in range.

GetInterfaceStatus (Method)

This method provides the status of the specified interface.

Inputs

Name	IEC 61131 Type	Description
interfaceIndex	UDINT	A numeric identifier of the interface. It must be greater than 0 and less than or equal to *NumInterfaces*.

Return Value

IEC 61131 Type	Description
enum_InterfaceStatus	The state of the requested interface.

Processing

- ➤ Check that *interfaceIndex* is within the specified range. Return NONE if it is not.
- ➤ Return the state of the specified interface.

GetInterfaceInfo (Method)

This method returns the information obtained from walking OID 1.3.6.1.2.1.2.2.1 before the *ENO* output was true.

NOTE: iso(1) . org(3) . dod(6) . internet(1) . mgmt(2) . mib-2(1) . interfaces(2) . ifTable(2) . ifEntry(1)

Inputs

Name	IEC 61131 Type	Description
interfaceIndex	UDINT	A numeric identifier of the interface. It must be greater than 0 and less than or equal to *NumInterfaces*.

Outputs

Name	IEC 61131 Type	Description
info	struct_InterfaceInfo	The structure containing the interface information.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the object is initialized and the *interfaceIndex* is in range.

Processing

- Check that *interfaceIndex* is within the specified range. Return FALSE if it is not, leaving the output in an undefined state.
- Copy internal information to the output and return true.

Reinitialize (Method)

Returns the agent to an uninitialized state, resulting in the *ENO* output being set to false. From this point, the agent automatically tries to re-initialize, refreshing all port information in the process.

Benchmarks

Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R134-V0 firmware
- SEL-3530-4
  - R133-V0 firmware
- SEL-3505
  - R133-V0 firmware

Benchmark Test Descriptions

class_SnmpManager.Run()

The posted time is the average execution time of 100 method calls when no traps are being received.

class_SnmpAgentMonitor.GetInterfaceStatus()

The posted time is the average execution time of 100 method calls.

class_SnmpAgentMonitor.GetInterfaceInfo()

The posted time is the average execution time of 100 method calls.

class_SnmpAgentMonitor.Reinitialize()

The posted time is the average execution time of 100 method calls.

class_SEL2730MAgent.GetPortInfo()

The posted time is the average execution time of 100 method calls.

class_SEL2730MAgent.GetInterfaceStatus()

The posted time is the average execution time of 100 method calls.

class_SEL2730MAgent.GetInterfaceInfo()

The posted time is the average execution time of 100 method calls.

class_SEL2730MAgent.Reinitialize()

The posted time is the average execution time of 100 method calls.

Benchmark Results

Operation Tested	**Platform (time in μs)**		
	SEL-3555	**SEL-3530-4**	**SEL-3505**
class_SnmpManager.Run()	22	409	644
class_SnmpAgentMonitor.GetInterfaceStatus()	1	11	49
class_SnmpAgentMonitor.GetInterfaceInfo()	1	4	92

Operation Tested	Platform (time in μs)		
	SEL-3555	SEL-3530-4	SEL-3505
class_SnmpAgentMonitor.Reinitialize()	1	65	12
class_SEL2730MAgent.GetPortInfo()	1	8	47
class_SEL2730MAgent.GetInterfaceStatus()	1	8	37
class_SEL2730MAgent.GetInterfaceInfo()	1	6	18
class_SEL2730MAgent.Reinitialize()	1	19	21

Examples

These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.

Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.

Monitoring Port Status of Three SEL-2730M Switches

Objective

A user has an SEL-RTAC that he wants to use to monitor link status of Ports 5–24 on three switches. He wants an alarm if one of those ports is inactive or if an additional port has a device plugged in.

Assumptions

- ➤ The RTAC being programmed can access three SEL-2730M devices configured on the network. These SEL-2730M devices have the following IP addresses:
    - ➢ 10.203.50.1
    - ➢ 10.203.50.101
    - ➢ 10.203.50.201
- ➤ The user has configured these switches to communicate with the RTAC via SNMP. This means the following:
    - ➢ The user has added a v2c profile to the switch with the community string "Switches" and at least Read permissions.
    - ➢ The user has added the RTAC to the switch as a trap server using the desired profile.
- ➤ The RTAC, configured as the SNMP manager, has an IP address 10.203.50.2.
- ➤ The Main task of the RTAC is not used for real-time critical purposes, making it appropriate to instantiate the SNMP manager object on the Main task.
- ➤ The RTAC has two access points configured to allow UDP traffic as follows:
    - ➢ UDP incoming on Port 161 for normal SNMP

➢ UDP incoming on Port 162 for SNMP traps

Solution

On the Main task, the user can create a program as shown in *Code Snippet 1*.

Code Snippet 1 prg_SnmpManagerWith3Agents

```
PROGRAM prg_SnmpManagerWith3Agents
VAR CONSTANT
    c_NumSwitches : UDINT := 3;
END_VAR
VAR
    SnmpManager : class_SnmpManager;
    Switch1 : class_SEL2730MAgent( iPAddress := '10.203.50.1',
                                   communityString := 'Switches');
    Switch2 : class_SEL2730MAgent( iPAddress := '10.203.85.101',
                                   communityString := 'Switches');
    Switch3 : class_SEL2730MAgent( iPAddress := '10.203.85.201',
                                   communityString := 'Switches');
    Alarm : BOOL;
    SwitchPointers : ARRAY[1..c_NumSwitches] OF POINTER TO
                     class_SEL2730MAgent := [ ADR(Switch1),
                     ADR(Switch2), ADR(Switch3)];
END_VAR
VAR_TEMP
    i, j : UDINT; // Iterators used.
END_VAR
```

```
// Run the SnmpManager, which is responsible for doing all the required
    work.
SnmpManager.Run();

// Initialize to success, so any unexpected result can flag failure.
Alarm := FALSE;
// Check that statuses match expected.
IF SwitchPointers[i]^.PortStatus[0] = SnmpLite.UP THEN
    // Alarm if something is plugged into the front port.
    // Note that ETHF status cannot be polled, so it will remain in
    // UNKNOWN state until a trap is received.
    Alarm := TRUE;
END_IF
FOR i := 1 TO c_NumSwitches DO
    FOR j := 1 TO 4 DO
        IF SwitchPointers[i]^.PortStatus[j] <> SnmpLite.DOWN THEN
            // Alarm if something plugged into the gigabit ports.
            Alarm := TRUE;
        END_IF
    END_FOR
    FOR j := 5 TO 24 DO
        IF SwitchPointers[i]^.PortStatus[j] <> SnmpLite.UP THEN
            // Alarm if something unplugged from ports 5-24.
            Alarm := TRUE;
        END_IF
    END_FOR
END_FOR
```

Release Notes

Version	Summary of Revisions	Date Code
3.5.1.0	➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types “Cannot convert” messages. ➤ Must be used with R143 firmware or later.	20180921
3.5.0.6	➤ Improved management of Ethernet socket failure.	20150511
3.5.0.5	➤ Removed limitation that number and SNMP ID of ports cannot change.	20150213
3.5.0.4	➤ Initial release.	20141010