California Polytechnic State University, San Luis Obispo

SNMP Integration into the CygNet SCADA System

In Satisfaction of the Senior Project Requirement

Paul Fake
6/10/2010

Table of Contents

Chapter 1: Introduction 4

Problem Statement 4

Project Stages 4

Chapter 2: Background 4

Simple Network Management Protocol 4

Management Information Base 5

Object Identifier 5

Versions 6

CygNet Software 6

Facilities 6

Points 7

Uniform Data Code 7

Remote Device EIE 7

Device Definition Service 7

Universal Interface Service 8

Chapter 3: Implementation Details 8

SNMP Library Selection 8

Raw Protocol 8

WinSNMP 8

Net-SNMP 8

NetSNMP EIE Development 10

UisNetSnmpDriver 10

CUisNetSnmpDriverDLLApp 12

CUisNetSnmpRemoteDevice 12

CUisNetSnmpRemoteMessageFactory 12
CGenericMsg and CSingleOidMsg 12
CxDdsNetSnmpEditors 12
CCxDdsNetSnmpEditor 13
CDdsNetSnmpDevPropPage 13
SNMP Integration 13
Creating a Device Template 14
Configuring the Device 15
Retrieving Data 15
Tabular MIB Support 17
Single OID Support 18
Testing 19
Chapter 4: Conclusion 20

Chapter 1: Introduction

In order to satisfy Cal Poly's Senior Project requirement, I have developed a software device that collects data from network-attached hardware using the Simple Network Management Protocol (SNMP). This software interfaces network-attached hardware with the project's client's data acquisition system, creating a useful tool for IT data analysis and monitoring. The client for this project is CygNet Software, Inc., a San Luis Obispo-based company that develops Supervisory Control and Data Acquisition (SCADA) software for oil well and gas pipeline companies. Part of their standard product consists of software devices, called "EIEs", which facilitate communication with remote hardware. The data from the hardware (typically meters attached to pipelines, which measure flow rate, temperature, gas composition, etc.) is collected into the SCADA system, and can be tracked in "Points" to be graphed and/or alarmed accordingly. I used this EIE model to create an SNMP device that collects management data from any type of hardware device on the network that supports SNMP. The device collects this data into CygNet Points and allows the tracking and graphing of historical data.

Problem Statement

IT departments in business enterprises tend to rely on third-party SNMP client software to remotely monitor routers, UPS's, CPU's, hard drives, and other hardware on the enterprise's network. This software may be expensive, depending on the quality, and it will have a learning curve. Enterprises running the CygNet SCADA system will be able to easily collect, store, and view SNMP data without the need to purchase and learn separate software.

Project Stages

The project consisted of 4 stages: research, SNMP library selection, CygNet EIE (the software device) development, and SNMP integration. In the research stage, I learned the SNMP protocol, as well as the CygNet EIE Remote Device model. In the SNMP library selection stage, I developed a test application to determine the best SNMP library to use. In the final two stages, I integrated the Net-SNMP library with the CygNet SCADA system via a Remote Device EIE.

Chapter 2: Background

Simple Network Management Protocol

SNMP is an application-layer protocol that utilizes the User Datagram Protocol (UDP) for communication [1]. SNMP is used to extract all sorts of diagnostic information from managed hardware. For example, one may use SNMP to monitor the battery levels of a UPS, or the disk usage of a server (referred to as "managed objects", or just "objects" for short). The protocol requires SNMP agent software to be running on the managed device from which data will be extracted, referred to here as the host. The SNMP agent stores all of its data into a tree of Management Information Bases (MIBs), which contain

variables that hold the numerical or text values representing system information. In order to request data, the client must send to the host an SNMP packet containing a path to the appropriate MIB, called an Object Identifier (OID), as well as the op code (which is SNMP_GET for the purposes of this project) [3].

Management Information Base

MIBs are text files representing collections of information about the objects which a host monitors. A MIB file contains individual entries for each MIB object in that particular set of data. The entry for the sysUpTime object in its MIB file looks like this:

```
sysUpTime OBJECT-TYPE
        SYNTAX  TimeTicks
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "The time (in hundredths of a second) since the
            network management portion of the system was last
            re-initialized."
        ::= { system 3 }
```

Figure 1: Example MIB object [1]

OBJECT-TYPE is a human-readable name (here, "sysUpTime") for an object. Aside from describing an object in a concise way, the OBJECT-TYPE string assists the client in translating this human-readable name to an OID. This is described in detail in the next section. SYNTAX tells the client how to interpret the data returned from the host. Here, TimeTicks means that the data will be a bit string representing a count of deci-seconds [3]. Other common SYNTAX types are "Integer" and "DisplayString." ACCESS tells the client whether or not the object can be manipulated. For our purposes we can just ignore this, since the project is only concerned with reading data from the host. STATUS is equally unimportant, so we can ignore it as well. DESCRIPTION is what it looks like – a more detailed description of the object than OBJECT-TYPE. The last line is a short-hand way of representing the object's OID. From this we can determine that the object is the third entry in the "system" branch [1].

A source of some confusion with SNMP is that a MIB, as defined above, is different from the actual database of values maintained by the agent, which is called the "MIB tree." The MIB tree is where the agent goes to look up the current value associated with a particular object when given an OID.

Object Identifier

An OID string is simply a unique string of integers representing an object's location in the MIB tree [1]. For example, the OID which retrieves the system's description is ".1.3.6.1.2.1.1.1.0". Each integer in the string specifies the branch to take at a given level of the tree to get to the desired object. Spelled out with the textual names of the branches, this OID would look like: iso.org.dod.internet.mgmt.mib-

2.system.sysDescr.0 [1]. The vast majority of objects lies in the (root).iso.standard.org.internet branch, and will therefore begin with .1.3.6.1.

As alluded to above, MIBs have a secret job of making object lookup easy by providing "nice-name translations." It would be inconvenient to have to type (and memorize!) a long string of integers to retrieve a simple object like a system description. So, nice SNMP clients and libraries will search through a local MIB directory in order to map a terminating leaf MIB name to an entire OID string. For example, the nice-name OID of .1.3.6.1.2.1.1.1.0 is simply "sysDescr.0."

Notice the ".0" at the end of the OID string, which does not have its own branch name. Each OID will end in an integer value, which specifies whether the object at that node is a *scalar* value or a table of objects. All scalar nodes are singular objects, and will end in ".0." Tabular objects will end in a number greater than zero, which specifies the column of data requested in the table [2]. The table entry (or row) is specified directly before the column, and varies based on the type of table. A good example of this is the ipRouteTable node. This table contains all of the information about IP routing that the system has collected. In order to retrieve the next hop IP address for 74.125.19.104 (a Google server), assuming this entry actually exists in the host's IP routing table, one must use the OID .1.3.6.1.2.1.4.21.74.125.19.104.7. The first part of the string, .1.3.6.1.2.1.4.21, is the location of the ipRouteTable branch, which is followed by the entry, 74.125.19.104, and lastly by the location of the data in the ipRouteNextHop column, .7. Some tables are static; that is, they do not have mutable entries keyed by complex strings like IP addresses. These tables can be thought of as simple groups of related ordinalized entries [2].

Versions

There are three versions of SNMP: SNMPv1, SNMPv2c, and SNMPv3. The main differences between the versions are mostly related to security implementation [4]. As most managed devices still use SNMPv1 [1], this project is limited to supporting only SNMPv1. The security model of SNMPv1 is very simple. Agents are given a "community name," which acts as a password for any incoming requests. All requests must contain the correct community name in order for the request to be accepted by the agent [1].

CygNet Software

In order to understand how the project integrates SNMP with CygNet, it is important to first understand some key components of the CygNet SCADA data model, namely facilities, points, Uniform Data Codes (UDCs), Remote Device EIEs, the Device Definition Service (DDS), and the Universal Interface Service (UIS).

Facilities

In CygNet, a facility is a logical grouping of data that represents a thing such as a gas measuring device, which can be modeled as a collection of identification, status, and measurement data. For our

purposes, a facility is a representation of a managed network device. A router, for example, might be characterized by its device name, network name, IP routing tables, etc. For this project, each of the data elements of a managed device facility is mapped to an OID.

Points

CygNet points represent the values for data elements in facilities. A point has one of four possible data types: analog, digital, string, and enumeration. Points can be configured to trigger alarms based on data values and trends. A point's record contains current value and history, current alarm state and history, and various configuration parameters, including associated facility and UDC.

Uniform Data Code

In order to associate a point with a facility's data element, the two must be tied together with a UDC, which is a standardized 10-character name. A UDC is mapped to a facility's data element, and a point is configured and resolved with both the UDC and facility name (in other words, a point can be uniquely identified by its UDC and facility). A single UDC can be created to represent the same type of data on multiple facilities. For example, a single UDC can be created to name CPU usage data, and every server on the network can associate this UDC with the appropriate data element.

Remote Device EIE

In CygNet, a Remote Device EIE (or "Equipment Interface Engine") is a driver that can communicate with a specific model of hardware. This driver knows the protocol and data format of a single device. The Remote Device model makes sense for managed network devices, as all network devices can be thought of as generic devices that communicate using SNMP. Thus, the Remote Device EIE created for this project (the NetSNMP EIE) is a driver that facilitates communication with any device that speaks SNMP.

A Remote Device instance is a software representation of a physical device which uses the EIE driver for communication. For all practical purposes, "facility" and "Remote Device instance" are synonymous. A Remote Device's data elements are configured via device templates (.dtf files). Templates are XML files which contain device definition data, such as device model name and type (the "type" of our EIE being "NetSNMP EIE"), and data groups. Each data group contains some number of data elements, referred to as Data Element IDs, or DEIDs. A DEID contains any information necessary to map the logical data element to an actual hardware data item, such as byte order, bit mask, and scale factors. For our purposes, a single Remote Device instance can be created to represent an entire multi-layered system such as a PC, with separate data groups for each piece of individual hardware on the system, or a separate Remote Device can be created for each part of the system, each with only one data group.

Device Definition Service

The DDS stores the configuration records for all Remote Device instances, and associates directly with a single UIS service.

Universal Interface Service

The UIS is the "workhorse" of CygNet. It hosts all of the EIE driver processes and all of the device instance threads, which in turn collect data for devices in the DDS, and stores records for points.

Chapter 3: Implementation Details

SNMP Library Selection

In order to begin integration of SNMP with CygNet, I needed a way to communicate with a managed device. I was faced with three options: raw protocol, the WinSNMP library, and the Net-SNMP library.

Raw Protocol

Implementing raw SNMP consists of writing software which constructs packets of data in the correct protocol format and sending them over the network to the host, waiting for a response, and parsing the returned packet. This method is the most complicated and time-consuming of the three, and I decided it would not be useful to re-write code that has already been written in SNMP libraries. Thus, I needed to decide which library best suited the project's needs.

WinSNMP

WinSNMP, part of the Windows API, is a Microsoft-developed SNMP library. Initially, WinSNMP was an attractive choice since CygNet is Windows-based, and using WinSNMP is as simple as including the correct header file as opposed to downloading and linking to an outside library. However, as I was becoming familiar with the WinSNMP library, I came to realize that the library did not have a concept of OID nice-name translation. That is, the string "sysDescr.0" was unrecognized by the API as a valid OID, so all requests had to be made with long-form OIDs. This turned out to be very inconvenient, as I, and any users of the software, would need to look up or memorize long OID strings for each requested object. Moreover, this API is actually fairly old and no longer updated, the MSDN documentation is confusing and thin, and there are very few examples of its use on the Internet. These shortcomings caused me to look elsewhere for my SNMP needs.

Net-SNMP

The open-source Net-SNMP library is the most widely-used library for SNMP development. I ultimately settled on this library because of its extensive documentation and the plethora of example code online, in addition to the lack of adequate alternatives. One downside of this library compared to WinSNMP is that it depends on outside source. My initial attempt to link dynamically to a DLL version of the library proved unsuccessful when I discovered that an author of the library redefined memory allocation functions (malloc(), realloc(), free(), etc.) to point to the Net-SNMP versions of these functions, which manage memory only in the DLL's memory space. What this means is that if memory is allocated in, for example, the UIS, and free() is called on that memory in a C++ file that includes a Net-SNMP header, the free() call will be reinterpreted as the special netsnmp_free(), causing a runtime memory exception. I

was able to get around this by downloading the source and linking to it statically, after fixing the redefines. The issue here is that the project now depends on a large amount of outside source instead of a simple DLL, increasing code size and reducing manageability, since updating the library is no longer as simple as downloading a new DLL and headers.

One requirement of the SNMP library was that it must be thread-safe, as the Net-SNMP EIE must allow multiple instances of Remote Devices to use the library at the same time. The Net-SNMP API is not inherently thread-safe, but it is possible to use it in a thread-safe manner. The README.thread document on the Net-SNMP website lists the following restrictions on the API's multi-threaded use [4]:

> 1. Invoke SOCK_STARTUP or SOCK_CLEANUP from the main thread only.
>
> 2. The MIB parsing functions use global shared data and are not
> multi-thread safe when the MIB tree is under construction.
> Once the tree is built, the data can be safely referenced from
> any thread. There is no provision for freeing the MIB tree.
> Suggestion: Read the MIB files before an SNMP session is created.
> This can be accomplished by invoking snmp_sess_init from the main
> thread and discarding the buffer which is initialised.
>
> 3. Invoke the SNMPv2p initialisation before an SNMP session is created,
> for reasons similar to reading the MIB file.
> The SNMPv2p structures should be available to all SNMP sessions.
> CAUTION: These structures have not been tested in a multi-threaded
> application.
>
> 4. Sessions created using the Single API do not interact with other
> SNMP sessions. If you choose to use Traditional API calls, call
> them from a single thread. The Library cannot reference an SNMP
> session using both Traditional and Single API calls.
>
> 5. Using the callback mechanism for asynchronous response PDUs
> requires additional caution in a multi-threaded application.
> This means a callback function probably should probably not use
> Single API calls to further process the session.
>
> 6. Each call to snmp_sess_open() creates an IDS. Only a call to
> snmp_sess_close() releases the resources used by the IDS.

Figure 2: Net-SNMP multi-threaded use restrictions [4]

These restrictions guided certain design decisions of the EIE, which will be discussed later. In order to test the multi-threaded capability of Net-SNMP, I developed a small test application, named SnmpTester. The tester is a Windows MFC Dialog application that requires a host name or IP, an OID in

long or nice-name format, and a community name from the user. When the user executes the request, 100 threads are launched, each of which sends an SNMP_GET request for the given OID. The results from the host are parsed and returned in an edit box.

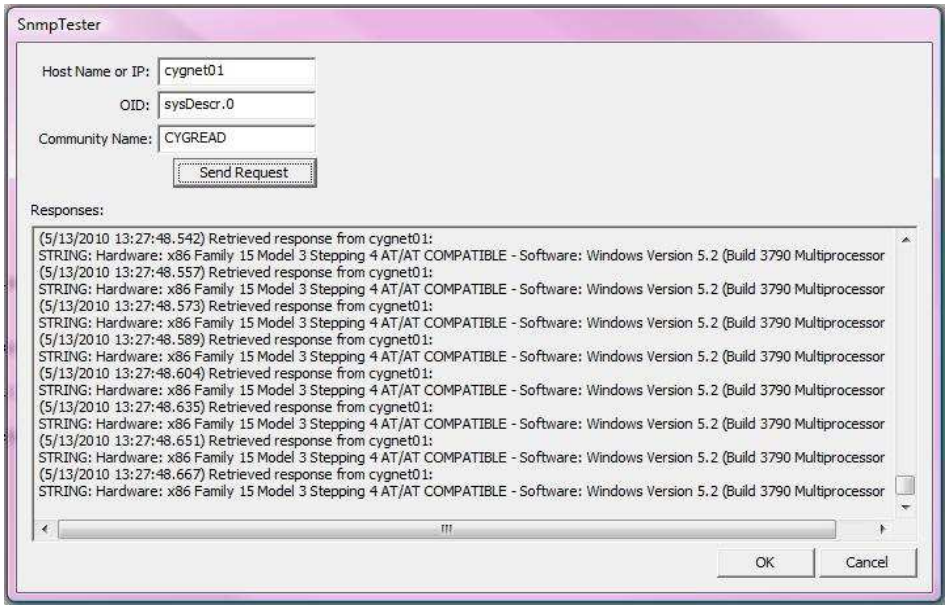

Figure 3: Screenshot of SnmpTester

SnmpTester successfully verified the thread-safety of the Net-SNMP API when the thread-safety guidelines are followed.

NetSNMP EIE Development

All CygNet EIEs have two main facets: the actual driver code, which communicates with the UIS, and the editor, which provides the user with an interface to create and manage devices in the DDS.

UisNetSnmpDriver

I developed the EIE driver by following the established EIE class hierarchy model, represented below in a UML diagram:

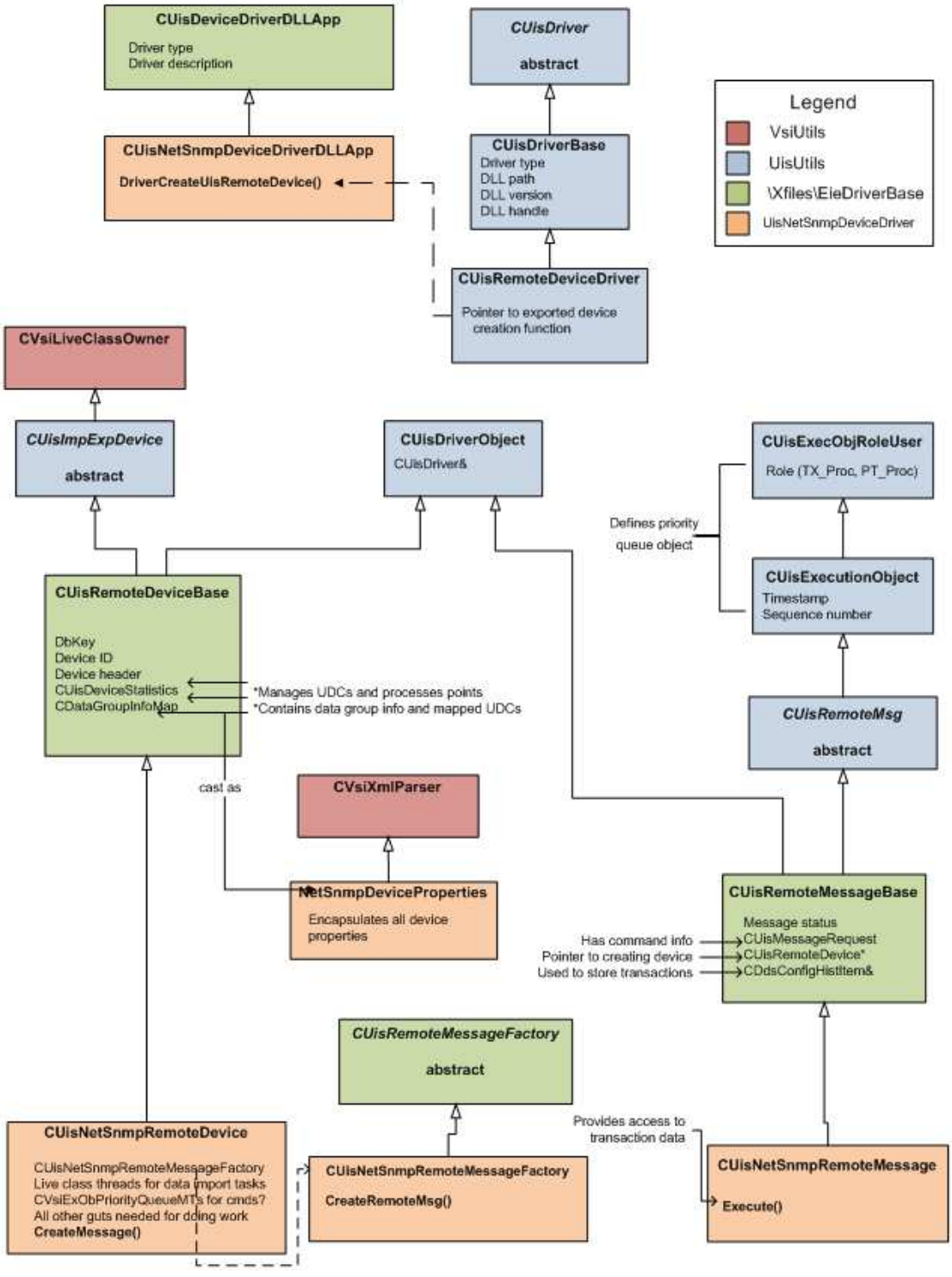

Figure 4: UML diagram for Remote Device EIE class hierarchy

CUisNetSnmpDriverDLLApp

The CUisNetSnmpDriverDLLApp is the top-level application class of the driver. This class is responsible for launching threads for device instances. In order to comply with guidelines 1 and 2 of the Net-SNMP multi-threading restrictions (see Figure 2), Net-SNMP initialization, cleanup, and MIB directory specification is done in this class. Since multiple device threads may try to add a MIB directory simultaneously, it was necessary to lock this operation with a busy-wait loop.

CUisNetSnmpRemoteDevice

A UisNetSnmpRemoteDevice object is the software encapsulation of a Remote Device instance. This class stores the device configuration from the DDS in a NetSnmpDeviceProperties object, and takes care of communication to a specific device via UisNetSnmpRemoteMessage objects. When the configuration for this device has been loaded (on UIS startup, or when changes are made to the device in the DDS), a session with the host device is established within this class.

CUisNetSnmpRemoteMessageFactory

The main purpose of this class is to handle all data requests from a Remote Device instance to be sent to a device. When data is requested, this class will generate a specific type of message based on the device's data group attributes (from the device template).

CGenericMsg and CSingleOidMsg

These two message classes, derived from CUisNetSnmpRemoteMessage, are responsible for the real work that is done to send SNMP requests to and receive data from a host. Ignoring for now the difference between a "generic" and a "single OID" type of device, these classes build an SNMP packet with the OID(s) specified in the device configuration, send the packet, wait for a response, parse the response, and update the point(s) associated with the OID(s).

CxDdsNetSnmpEditors

I developed the NetSNMP Remote Device editor by following the established DDS editor class hierarchy model, represented below in a UML diagram:

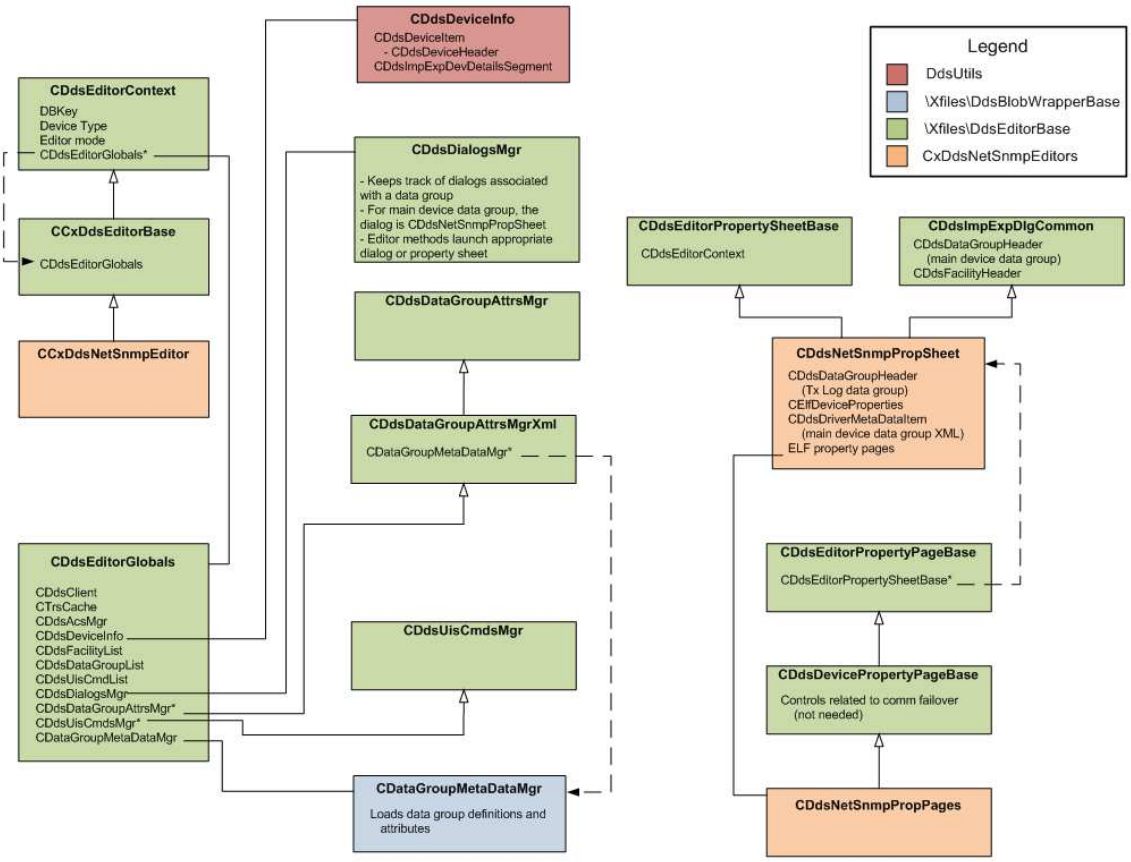

Figure 5: UML diagram for DDS editor class hierarchy

CCxDdsNetSnmpEditor

This is the top-level class responsible for loading the property sheet and dialogs when managing a Remote Device instance in the DDS.

CDdsNetSnmpDevPropPage

This class provides an interface between the user and the Remote Device instance in the DDS. It is derived from a Windows property page control, and it contains various editable fields relevant to the NetSNMP device (such as host IP and community name). This will be explored in greater detail in the next section.

SNMP Integration

With the EIE developed, the next task was to complete integration of SNMP into CygNet by creating a NetSNMP Remote Device instance and collecting data. Part of the requirement here was to support tabular MIBs, as well as a "Single OID" device model, but I will start by talking about the simplest use case, and I will address the other features later.

Creating a Device Template

Before creating a device, it was necessary to write a device template to set up the data elements for the specific type of device to be integrated. For this example, I created a template for a Remote Device which represents general system info for any managed device:

```
<deviceDefinition deviceType="NetSnmp" eieType="NetSnmp" category="4098" mfg="CygNet Software,
Inc." model="Net SNMP" desc="Net SNMP Template">
        <dataGroups udcCat="~UDCALL" canSend="false" canRecv="true" uccSend="false"
uccRecv="true" udcDefFac="true" devDG="false" baseOrd="0" maxCnt="1">
        <!-- common datagroup to retrieve device config information -->

    <SysInfo niceName="SNMP System Info" dgDesc="SNMP System Info">
        <dgElements>
            <Desc desc="Device Description" type="string" oid="sysDescr.0"/>
            <ObjectID desc="Device OID" type="string" oid="sysObjectID.0"/>
            <UpTime desc="Device Up Time" type="ui4" oid=".1.3.6.1.2.1.1.3.0"/>
            <UpTimeMs desc="Device Up Time (msecs)" type="r8" ref="UpTime" scaleFactor="10"
units="milliseconds"/>
            <UpTimeS desc="Device Up Time (secs)" type="r8" ref="UpTimeMs" units="seconds"/>
            <Contact desc="Device Contact" type="string" oid="sysContact.0"/>
            <Name desc="Device Name" type="string" oid="sysName.0"/>
            <Location desc="Device Location" type="string" oid="sysLocation.0"/>
            <Services desc="Device Services" type="ui4" oid="sysServices.0"/>
        </dgElements>
    </SysInfo>

</dataGroups>
<defUisCmds visible="true" canBeScheduled="true" clientCanInvoke="true" inheritsSecurity="false">
</defUisCmds>
</deviceDefinition>
```

Figure 6: NetSnmp.dtf

This template gives the device model a generic name of “Net SNMP” (which is admittedly not very creative or informative). The data group SysInfo groups all of the child elements together as a coherent set of data. Each DEID (Desc, ObjectID, etc.) represents a data element on the host, and is mapped to an OID (which can be specified in either long form or nice-name form). The “type” attribute tells the EIE how to interpret the data returned from the host. “String” is a simple string of characters, “ui4” is an unsigned 4-byte integer, and “r8” is an 8-byte double-precision floating-point value. One difficulty that came up at this point was “what do I do about the units of returned data?” As noted in a previous section, sysUpTime is measured in deciseconds, a very awkward unit of time. Furthermore, CygNet Points allow the user to specify whatever units of data he or she pleases, so it became necessary to support configurable units. This was accomplished through what is known as “Reference DEIDs.” Notice that the UpTimeMs and UpTimeS DEIDs do not map directly to their own OIDs. Rather, they have a special “ref” attribute, which ties them to another DEID. Referencing a DEID creates a pseudo-data element by taking the value from a data element and doing some work on it. In the UpTimeMs DEID, the “scaleFactor” attribute specifies the amount by which the UpTime value must be scaled to represent the desired units, and the “units” attribute lets the system know what that desired unit is. The UpTimeS data element references the UpTimeMs DEID and uses the units “seconds.” Since “seconds” and “milliseconds” are known to the CygNet system, it is not necessary to scale the values manually. Now that there are three separate data elements for UpTime, a Point can be created for each one using different units.

Configuring the Device

Using CygNet's proprietary Explorer program, CExplore, I created the Remote Device instance in the DDS using the template in Figure 6. The following property sheet developed in CxDdsNetSnmpEditors is what appears as the first step of Remote Device configuration:

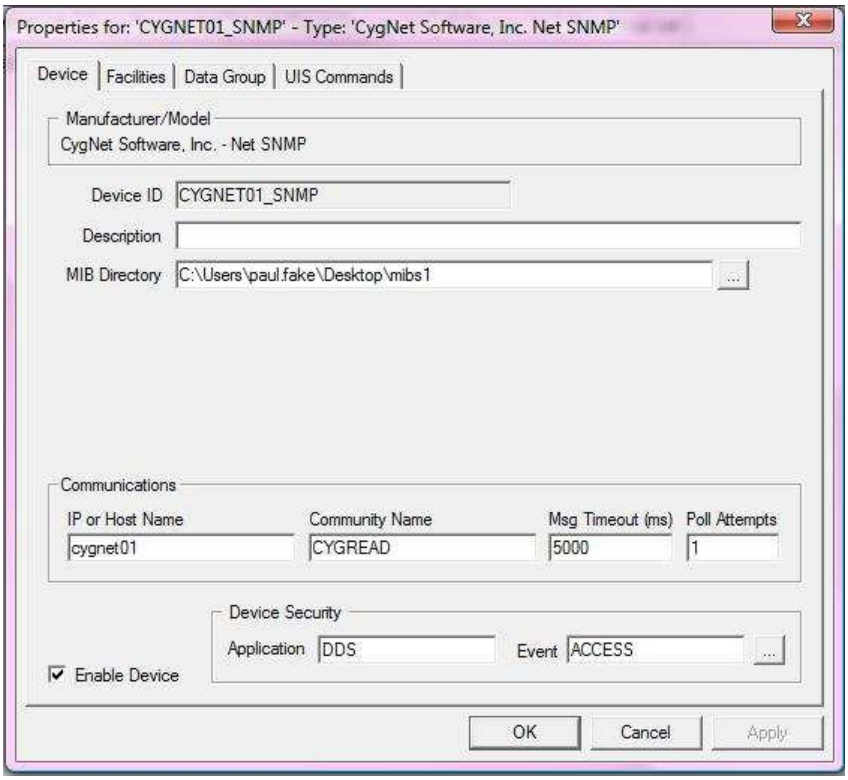

Figure 7: Screenshot of NetSNMP Remote Device property sheet

This first tab allows the user to specify all information required to communicate with a managed device. The "IP or Host Name" in this example is configured to be the host name of a Windows server on the network. The rest of the options in the "Communications" group are self-explanatory. The MIB Directory option specifies the directory in which the local machine will search for a nice-name OID translation. If no MIB directory is specified, the Net-SNMP API will not be able to resolve nice-name OIDs. The options in the "Device Security" group are CygNet-specific and are outside the scope of this paper. After the "IP or Host Name" and "Community Name" options are filled in, the user can press "Apply" and finally create the device.

Retrieving Data

The way to manually retrieve data from this device is to first navigate to the "Data Group" tab:

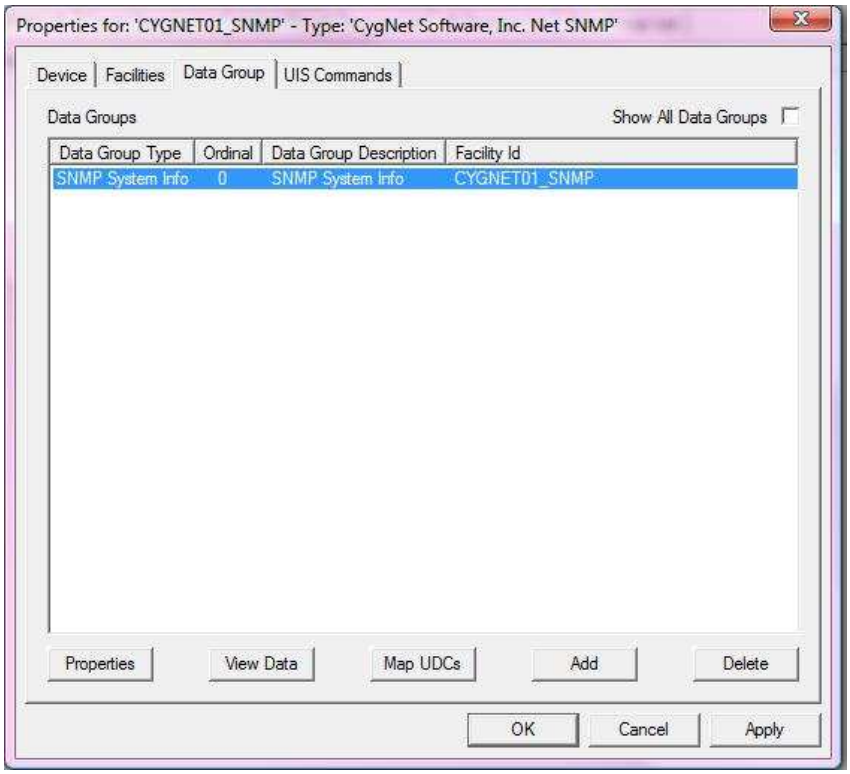

Figure 8: Screenshot of the "Data Group" tab of the NetSNMP Remote Device property sheet

The only data group featured is the one we configured in the device template. Selecting this data group and clicking on "View Data" brings us to the following dialog:

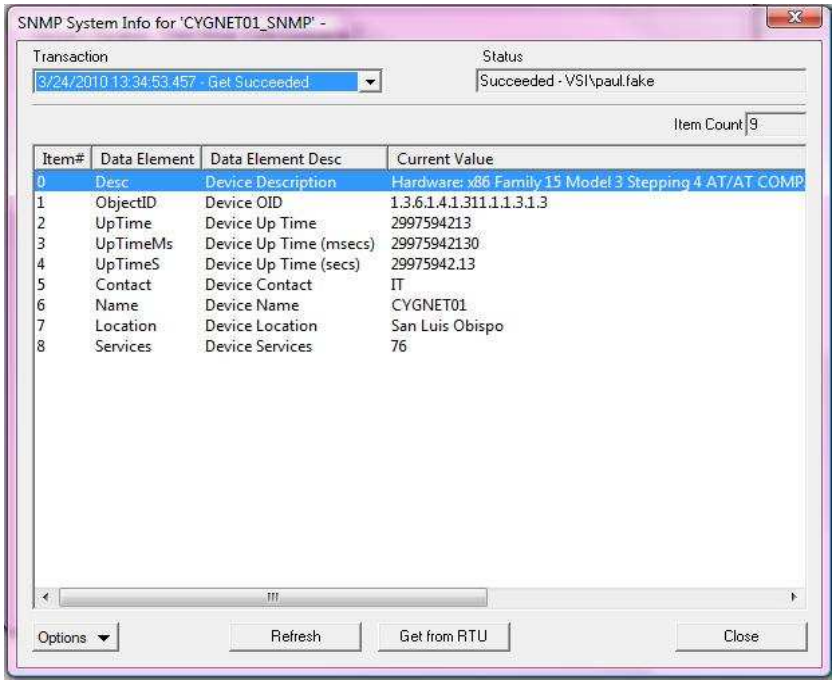

Figure 9: Screenshot of the "View Data" window

Clicking on the “Get from RTU” button sends a new SNMP request, and updates the listed values. All of the items in the list are the data elements we configured in the device template.

Another (and more useful) way to retrieve data is to map these data elements to Points (via UDCs), using the “Map UDCs” button shown in Figure 8. Once the data elements are mapped, one can use CygNet’s scheduling service to retrieve data at a given interval. The following is a screenshot of a graph of historical UpTime data for this device, scheduled to retrieve every two seconds:

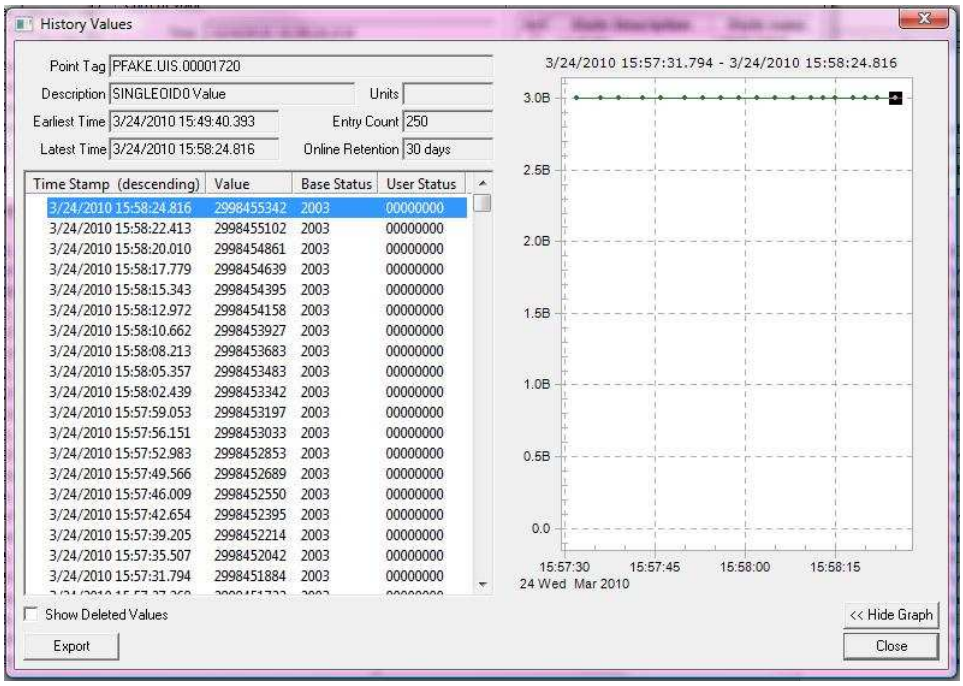

Figure 9: Screenshot of “History Values” view in the UIS

This isn’t the most interesting graph of data, but you can imagine what a historical graph of CPU usage might look like.

Tabular MIB Support

Support for tabular MIBs is accomplished through the device template:

```
<deviceDefinition deviceType="NetSnmpSysOR" eieType="NetSnmp" category="4098" mfg="CygNet
Software, Inc." model="Net SNMP" desc="Net SNMP SysORTable">
       <dataGroups udcCat="~UDCALL" canSend="false" canRecv="true" uccSend="false"
uccRecv="true" udcDefFac="true" devDG="false" baseOrd="1" maxCnt="5">
        <!-- common datagroup to retrieve device config information -->

    <SysORInfo niceName="SNMP System OR Info" dgDesc="SNMP System OR Info">
        <dgElements>
            <Desc desc="Entry Description" type="string" oid="sysORDescr.{ORD}"/>
            <ObjectID desc="Entry OID" type="string" oid="sysORID.{ORD}"/>
            <UpTime desc="Entry Up Time" type="ui4" oid="sysORUpTime.{ORD}"/>
            <UpTimeMs desc="Entry Up Time (msecs)" type="r8" ref="UpTime" scaleFactor="10"
units="milliseconds"/>
            <UpTimeS desc="Entry Up Time (secs)" type="r8" ref="UpTimeMs" units="seconds"/>
            <Index desc="Entry Index" type="string" oid="sysORIndex.{ORD}"/>
        </dgElements>
    </SysORInfo>
```

```
</dataGroups>
<defUisCmds visible="true" canBeScheduled="true" clientCanInvoke="true" inheritsSecurity="false">
</defUisCmds>
</deviceDefinition>
```

Figure 10: NetSnmp_SysORTable.dtf

In this template, the “oid” attributes do not explicitly specify the entry ordinal in the sysORTable. Instead, it uses the “{ORD}” substitution string to be replaced by the EIE with an ordinal that can be configured in the DDS after the Remote Device instance has been created.

Single OID Support

It may be useful to create a device that is not configured with a specific set of data elements, but rather the ability to retrieve data from any element on the fly. To accomplish this, I have created the “Single OID” category of devices. To create a Single OID type of device, a different style of device template must be used:

```
<deviceDefinition deviceType="NetSnmpSingleOID" eieType="NetSnmp" category="4098" mfg="CygNet
Software, Inc." model="Net SNMP" desc="Net SNMP Single OID">
       <dataGroups udcCat="~UDCALL" canSend="false" canRecv="true" uccSend="false"
uccRecv="true" udcDefFac="true" devDG="false" baseOrd="0" maxCnt="1">
        <!-- common datagroup to retrieve device config information -->

    <SysInfo niceName="SNMP Single OID" dgDesc="SNMP Single OID" dgCat="singleOid">
        <dgElements>
            <Oid desc="Object Identifier" type="string"/>
            <DataType desc="Data Type" type="string"/>
            <Value desc="Data Value" type="vrnt"/>
        </dgElements>
        <uccRecvParms>
            <Oid desc="Object Identifier" required="true" type="string"/>
            <DataType desc="Data Type" required="true" type="string"/>
        </uccRecvParms>
    </SysInfo>

</dataGroups>
<defUisCmds visible="true" canBeScheduled="true" clientCanInvoke="true" inheritsSecurity="false">
</defUisCmds>
</deviceDefinition>
```

Figure 11: NetSnmp_SingleOID.dtf

Instead of explicit data elements, the DEIDs in this template refer to the generic parameters of OID, returned value type, and value. “Oid” and “DataType” are filled in by the user prior to sending a request (which we will see shortly), and “Value” is filled in by the response with a variant type, meaning it will be interpreted as either a string or an integer depending on the value of “DataType.” The “uccRecvParms” elements define the parameters that must be supplied to the request. Retrieving data in Single OID mode is a little different from the previous methods. First of all, clicking on “View Data” in the “Data Groups” tab (see Figure 8) will now bring up a different dialog:

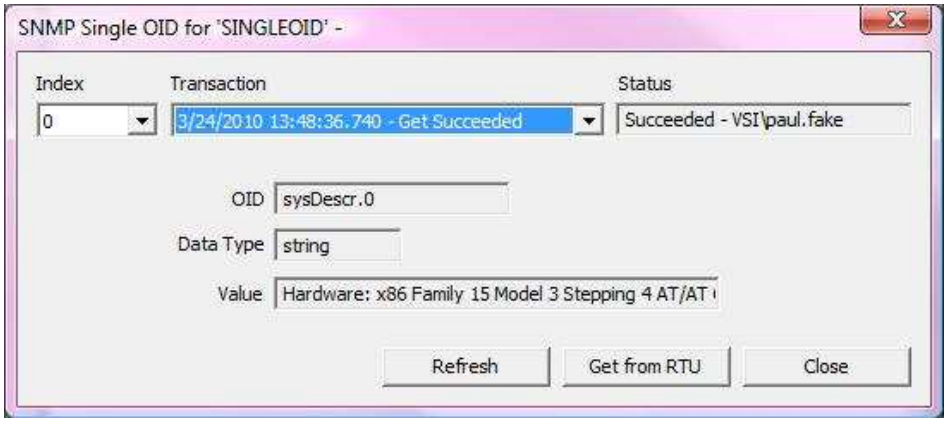

Figure 12: Screenshot of the "View Data" window for Single OID mode

Clicking on "Get from RTU" brings up a dialog that allows the user to enter the required request parameters:

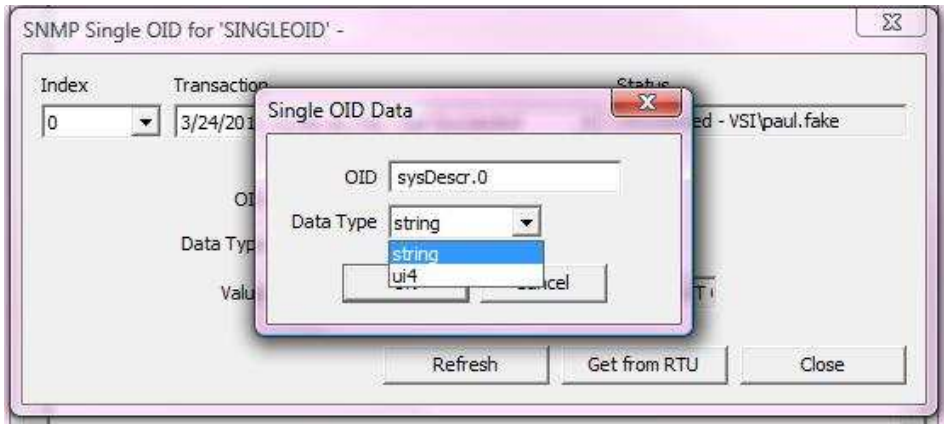

Figure 13: Screen of the "Get from RTU" window for Single OID mode

Scheduling of data is possible as it was previously, as scheduled commands to the UIS can be configured with the same request parameters.

Testing

In order to verify the thread-safety of Remote Device instances, I created 100 Single OID Remote Device instances, and wrote a VB Script file that requests 1000 data points from each simultaneously. All data was retrieved and stored in the UIS successfully.

Chapter 4: Conclusions and Future Work

The project described by this paper is in satisfaction of the criteria for a CPE Senior Project. Development of the SNMP Remote Device required substantial research into the SNMP protocol and the Net-SNMP library, as well as creativity in order to integrate the protocol into CygNet in a seamless and intuitive way. Lastly, the project integrates software (the Remote Device) and hardware (the actual managed network devices) with communications over a network protocol, drawing upon knowledge from CPE 464: Introduction to Computer Networks.

Overall, the project was a success. I met all of the requirements proposed by the client, integrating managed network device information into the CygNet system. One element missing from the project, however, is explicit support for non-ordinalized tabular MIBS, such as the ipRouteTable. The entries in these tables are accessed not by simple integers, but rather by complex strings of numbers such as an IP address. It is still possible to retrieve data from these tables by using the Single OID model or, worse, explicitly naming the entry in the template file, but I could not figure out a good model for creating this sort of device generically. Fortunately, the client did not require this to be done for the project, but it is a potential item to be included in a future release. I am satisfied with the work I have done, and I have learned a lot about SNMP and the CygNet system.

Project Status

Currently, there are no plans for CygNet to either release the SNMP Remote Device to customers or use it internally. CygNet has expressed interest in the further development of the device for future use, but no requirements have yet been made.

References

[1] SUNY Institute of Technology. "SNMP for Dummies." *The CompuTech Group*. WordPress, 25 May 1998. Web. 5 May 2010. <http://www.computechgroup.com/?p=31>.

This document provides an introduction to the various aspects of SNMP. It functions as a starting point to learning about MIBs and OIDs.

[2] Case, J., et al. "RFC 1157." *A Simple Network Management Protocol (SNMP)*. Internet Engineering Task Force, May 1990. Web. 7 Jun. 2010.<http://www.ietf.org/rfc/rfc1157.txt>.

This RFC defines the SNMP protocol.

[3] Case, J., et al. "RFC 1902." *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*. Internet Engineering Task Force, Jan. 1996. Web. 7 Jun. 2010. <http://tools.ietf.org/html/rfc1902>.

This RFC defines the structure of MIBs, as well as the different types of variables contained in these structures.

[4] Case, J., et al. "RFC 2570." *Introduction to Version 3 of the Internet-standard Network Management Framework*. Internet Engineering Task Force, Apr. 1999. Web. 7 Jun. 2010. <http://tools.ietf.org/html/rfc2570>.

This RFC provides an overview of the third version of SNMP. It outlines the major differences between the three version.

[5] "README.thread." *Net-SNMP*. 2 Mar. 2007. Web. 25 Feb. 2010. <http://net-snmp.sourceforge.net/docs/README.thread.html>.

This README contains information on the thread-safety of the Net-SNMP API. It details why the API is not natively thread-safe, but it gives information about how to use it in a thread-safe manner.

[6] "Net-SNMP." *Net-SNMP*. 2 Mar. 2007. Web. 25 Feb. 2010. <http://www.net-snmp.org/>.

I downloaded the Net-SNMP API and agent here. I also used the tutorial section to develop a test application using the API.

[7] "WinSNMP API (Windows)." *MSDN: Microsoft Development, MSDN Subscriptions, Resources, and More*. Microsoft, 19 Nov. 2009. Web. 25 Feb. 2010. <http://msdn.microsoft.com/en-us/library/aa379207%28VS.85%29.aspx>.

This MSDN article is the official documentation for the WinSnmp API. It contains (very) brief descriptions of functions.