SNMP++

An Object Oriented Approach
For Network Management
Programming
Using C++

Revision 2.1

Peter E Mellquist
Network Management
Roseville Networks Division
Hewlett Packard Company

Technical Contributors:
Kim Banker
Gary Berard
Chuck Black
Bruce Falzarano
Harry Kellog
Moises Medina
Tom Milner
Mark Pearson

Table Of Contents

What's New in Revision 2.0

SNMP++ revision 2.0 includes a variety of new enhancements. Enhancements include new features, increased flexibility and better performance. The following is a summary of new features for rev 2.0.

- **Asynchronous SNMP requests for MS-Windows**
  SNMP++ for MS-Windows now supports both blocked and non-blocked (asynchronous) modes. The UNIX implementation does not currently support async mode.

- **Rendezvous shutdown mechanism**
  Rendezvous shut down mechanisms for globally or partially shutting down a blocked SNMP++ request is now possible.

- **Medina's many engine**
  Medina's Many Engine is a powerful member function for obtaining SNMP objects in bulk. The m*any engine* makes it easy for the implementor to grab up to fifty objects from a device in one call . As SNMP++ migrates to SNMP v2, the internals of the *many engine* will utilize SNMP version 2's get-bulk.

- **Timeticks, Counter and Gauge objects**
  These three new SNMP++ classes make getting and setting SMI TimeTicks, Counters and Gauge Objects easy.

- **New Oid class member functions**
  A variety of new Oid class member functions were created extending the functionality and power of the Oid class.

- **Medium memory model support for MS-Windows**
  SNMP++ may now be compiled in the MS-Windows medium or large model.

- **Windows NT and Windows '95 Beta Support**
  A Win16 SNMP++ application will now function under Win32 driving through WinSNMP and NT's WinSock protocol stack. This includes Windows '95 Beta II.

- **Trap Support for MS-Windows**
  SNMP++ now includes support for arming and receiving traps for MS-Windows.

- **HPUX support for series 700 and 800 workstations**
  An SNMP++ HPUX app can now operate on series 700 or 800 HP Workstations.

- **Extended Error Codes**
  SNMP++ error codes have been extended to provide more detail on possible errors which can occur.

- **Faster, More Efficient Oid Class**
  Leaner and faster Oid class offers significant performance improvements.

- **Runs over FTP's and ACEC (American Computer Electronics Corp) WinSNMP DLL**
  SNMP++ has been tested over FTP Softwares and ACEC s NetPlus WinSNMP

Products Now Using SNMP++

- **HP DownLoad Manager For MS-Windows**
  Uses SNMP++ for MS-Windows running over WinSNMP. Runs over IP & IPX on MS-Windows 3.1, Windows For Work Groups 3.11(WFWG) and Windows NT.

- **HP DownLoad Manager For HPUX**
  Uses SNMP++ for HPUX. Runs over IP on series 700 and 800 HP work stations.

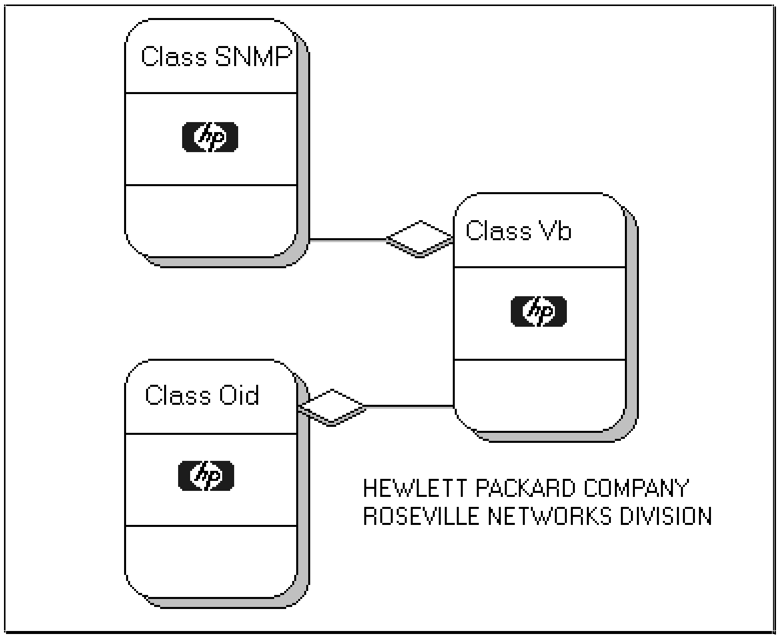
- **HP Router Monitor For MS-Windows**
  Uses SNMP++ for MS-Windows running over WinSnmp. Runs over IP on MS-Windows 3.1, WFWG and Windows NT for the HP OpenView for Windows platform.

- **HP Router Monitor For HPUX**
  Uses SNMP++ for MS-Windows running over WinSnmp. Runs Over IP on series 700 and 800 HP workstations for the HP OpenView for HPUX platform.

- **HP InterConnect Manager (ICM) For MS-Windows**
  Uses SNMP++ for MS-Windows running over WinSnmp. Runs over IP on MS-Windows 3.1, WFWG and Windows NT. Operates in a stand alone manner or with HP OpenView for Windows.

- **HP InterConnect Manager For HPUX**
  Uses SNMP++ for MS-Windows running over WinSnmp. Runs Over IP on series 700 and 800 HP workstations for the HP OpenView for HPUX platform.

- **SNMP++ Demo Application**
  A powerful browser application which demonstrates the ease, power and flexibility of SNMP++. Implemented using MS-Visual C++ and MFC for Win16. The demo application is available through HP Roseville Network Division for evaluation purposes.



Introduction

Various Simple Network Management Protocol (SNMP) Application Programmers Interfaces (APIs) exist which allow for the creation of network management applications. The majority of these APIs provide a large library of functions which require the programmer to be familiar with the inner workings of SNMP and SNMP resource management. Most of these APIs are platform specific, resulting in SNMP code specific to an operating system or network operating system platform and thus not portable. Application development using C++ has entered the main stream and with it a rich set of reusable class libraries are now readily available. What is missing is a standard set of C++ classes for network management. An object oriented approach to SNMP network programming provides many benefits including ease of use, safety, portability and extensibility. SNMP++ offers power and flexibility which would otherwise be difficult to implement and manage.

What Is SNMP++

SNMP++ is a set of C++ classes which provide SNMP services to a network management application developer. SNMP++ is not an additional layer or wrapper over existing SNMP engines. SNMP++ layers over existing SNMP libraries in a few minimized areas and in doing so is efficient and portable. The majority of SNMP++ includes a full implementation of SNMP. SNMP++ is not meant to replace other existing SNMP APIs such as WinSNMP, rather it offers power and flexibility which would otherwise be difficult to manage and implement. SNMP++ brings the ***Object Advantage*** to network management programming.

SNMP++ Objectives

Ease of Use

An Object Oriented (OO) approach to SNMP programming should be easy to use. After all, this is supposed to be a *simple* network management protocol. SNMP++ attempts to put the *simple* back into SNMP! The application programmer does not need be concerned with low level SNMP mechanisms. An OO approach to SNMP encapsulates and hides the internal mechanisms of SNMP. This provides safety since it protects the programmer from inadvertently doing the wrong thing. In regard to ease of use, SNMP++ addresses the following areas.

Provides an easy-to-use interface into SNMP

A user does not have to be an expert in SNMP to use SNMP++. Furthermore, a user does not have to be an expert in C++!

Preserves the flexibility of lower level SNMP programming

A user may want to bypass the OO approach and code directly to low level SNMP calls. SNMP++ is fast and efficient. However, there may be instances where the programmer requires coding directly to an SNMP API.

Encourage programmers to use the full power of C++ without chastising them for not learning fast enough

A user does not have to be an expert in C++ to use SNMP++. Basic knowledge of SNMP is required, but as will be shown, a minimal understanding of C++ is needed.

Safety

Most SNMP APIs require the programmer to manage a variety of resources. These include Object Id's (Oids), Variable Bindings (Vbs), Variable Binding Lists (Vbls), Protocol Data Units (PDUs), Community Names, and authentication structures [RFC 1442]. Improper allocation or de-allocation of these resources can result in corrupted or lost memory. SNMP++ provides safety by managing these resources internally. The user of SNMP++ realizes the benefits of automatic resource and session management. In regard to safety, SNMP++ addresses the following areas.

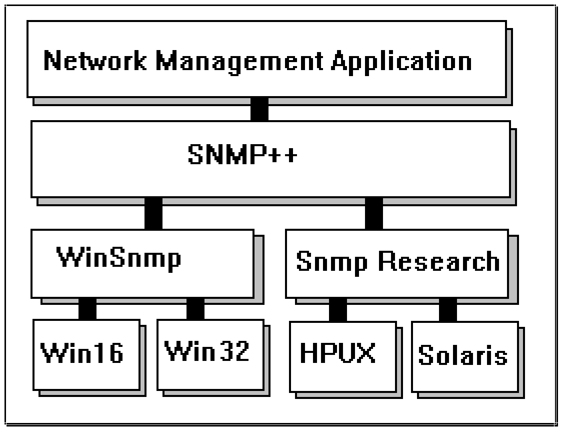
Provides automatic management of SNMP resources.

This includes SNMP structures, sessions, and transport layer management.
SNMP classes are designed as Abstract Data Types (ADTs) [Saks]. This includes data hiding and the provision of public member functions to inspect or modify hidden instance variables.

Provides built in error checking, automatic timeout and retry

A user of SNMP++ does not have to be concerned with providing reliability for an unreliable transport mechanism. SNMP relies on network transport layer communication via unreliable services (eg. UDP , IPX) [Stallings]. A variety of communications errors can occur including: lost datagrams, duplicated datagrams, and reordered datagrams. SNMP++ addresses each of these possible error conditions and provides the user with transparent reliability.

Portability



A major goal of SNMP++ is to provide a portable API across a variety of operating systems (OSs), network operating systems (NOSs), and network management platforms. Since the internal mechanisms of SNMP++ are hidden, the public interface remains the same across any platform. *A programmer who codes to SNMP++ does not have to make changes to move it to another platform.* The current working SNMP++ platforms include MS-Windows 3.1, MS-Windows For Work Groups 3.11, MS-Windows NT, MS-Windows '95 Beta II, and HPUX (HP UNIX). *Note!, Currently only Win16 is supported.* Platforms currently supported are HP OpenView for Windows and HP OpenView for HPUX. Another issue in the area of portability is the ability to run across a variety of protocols. SNMP++ currently operates over the Internet Protocol (IP) or Internet Packet Exchange (IPX) protocols, or both using a dual stack.

Extensibility

Extensibility is not a binary function but rather one of degree. SNMP++ not only can be extended, but can and has been extended easily. Extensions to SNMP++ include supporting new OS's, NOS's , network management platforms, protocols, supporting SNMP version 2, and adding new features. Through C++ class derivation, users of SNMP++ can inherit what they like and overload what they wish to redefine.

OverLoading SNMP++ Base Classes

The application programmer may subclass the base SNMP++ classes to provide specialized behavior and attributes. This theme is central to object orientation [Gama]. The base classes of SNMP++ are meant to be generic and do not contain any vendor specific data structures or behavior. New attributes can be easily added through C++ sub-classing and member function redefinition.

An Introductory Example

Rather than begin by describing SNMP++ and all of its features, here is a simple example that illustrates its power and simplicity. The following example is designed to run on MS-Windows 3.1 using the 3.1 API [Petzold]. This example obtains a System Descriptor object from the specified agent. Included is all code needed to create a session, get an SMI octet variable, and print it out. Retries and time-outs are managed automatically. The SNMP++ code is in bold font.

Windows 3.1 Example

```
#include "snmp.h"
void get_system_descriptor( HWND hWnd)
{
  int status;
  long int err_status, err_index;
  char msg[255];
  Vb vb;                              // construct a vb object
  vb.set_oid("1.3.6.1.2.1.1.0");     // get the system descriptor
  // construct a Snmp Object
  Snmp snmp( hWnd, (Protocol) ip, "public", &status);   // construct an ip snmp object
  if ( status != SNMP_CLASS_SUCCESS)
  {
    MessageBox( hWnd,"Failure Instantiating SNMP Class!","Snmp++ Error",MB_ICONSTOP);
    return;
  }

  snmp.set_retry(3);     // set retries @ 3, default is 1 second
  status = snmp.get(&vb,1,err_status, err_index,"15.29.33.10");   // get the data
  if ( status != SNMP_CLASS_SUCCESS)
  {
   sprintf(msg,"Get Fail %d ",status);
   MessageBox( hWnd,msg,"Snmp++ Error",MB_ICONSTOP);  // display it
  }
  else
  {
   vb.get_value( (char *)msg);      // extract the char string into msg
   MessageBox( hWnd,msg,"System Descriptor", MB_OK);
  }
};
```

Explanation of Introductory Example

The majority of code in the above example provides for error checking . The actual SNMP++ calls are made up of six lines of code. Two SNMP++ objects are utilized, the Variable Binding (Vb) object and the SNMP object. The Vb object is constructed and two public member functions are utilized. Vb::set_oid, sets the Oid portion of the Vb object. Vb::get_value extracts a octet array from the returned Vb object.

SNMP++ Features

Oid, Vb and SNMP Objects

SNMP++ is based around three C++ classes, the SNMP Object Identification (OID) class, the SNMP Variable Binding (Vb) class, and the SNMP class. Together, these classes give the programmer full SNMP management support.

Automatic SNMP Resource Memory Management

The Oid, Vb and SNMP classes manage various SNMP structures and resources automatically when objects are instantiated and destroyed. This frees the application programmer from having to worry about de-allocating structures and resources and thus provides better protection from memory corruption and leaks. SNMP++ objects may be instantiated statically or dynamically. Static object instantiation allows destruction when the object goes out of scope. Dynamic allocation requires use of C++ constructs *New* and *Delete* [Stroustrup]. Internal to SNMP++ are various Structure of Management Information (SMI) structures which are protected and hidden from the public interface. All SMI structures are managed internally, the programmer does not need to define or manage SMI structures or values.

Ease Of Use

By hiding and managing all SMI structures and values, the SNMP++ classes are easy and safe to use. The programmer cannot corrupt what is hidden and protected from scope.

Power and Flexibility

SNMP++ provides power and flexibility which would otherwise be difficult to implement and manage. Each SNMP++ object communicates with an agent through a session model. That is, an instance of a SNMP++ class maintains a connection to the specified agent. Each SNMP++ object provides reliability through automatic retry and timeouts. An application may have multiple SNMP++ object instances, each instance communicating to the same or different agent(s). This is a powerful feature which allows a network management application to have different sessions for each management component. For example, an application may have one SNMP++ object to provide graphing statistics, another SNMP++ object to monitor traps, and a third SNMP++ object to allow SNMP browsing. SNMP++ automatically handles multiple concurrent requests from different SNMP++ instances.

Portable Objects

The majority of SNMP++ is portable C++ code. This includes the Oid and Vb classes. The SNMP class definition is portable as well. Only the SNMP class implementation is different for each target operating system. *If your program contains SNMP++ code, this code will port without any changes!*

Automatic Timeout And Retries

SNMP++ supports automatic timeout and retries. This frees the programmer from having to implement timeout or retry code. The SNMP class supports two public member functions for accessing and modifying the retry and timeout behavior. Automatic timeout and retry is exclusive to blocked mode SNMP++ objects.

Blocked Mode Requests

SNMP++ includes a blocked model. The blocked model supported allows multiple blocked requests on separate SNMP class instances. The blocked model provides a cleaner, simpler SNMP interface while introducing no restrictions. *Note!, blocked mode only applies to individual SNMP object instances. You may have multiple instances which operate asynchronously.*

Non-Blocking Asynchronous Mode Requests

For the MS-Windows environment, SNMP++ supports a non-blocking asynchronous mode for gets, sets and get-nexts. For this mode of operation, the programmer is responsible for handling time-outs and retries. Asynchronous mode lends itself well for applications which do periodic polling such as graphing.

Traps

For the MS-Windows environment, SNMP++ supports trap reception. Traps are received through WinSNMP which manages the well known UDP or IPX trap port. This allows an SNMP++ application to coexist with other applications receiving traps on the same computer. This is the case with applications wishing to coexist with HP OpenView for MS-Windows.

Support For SNMP Version 1

The current implementation for SNMP++ is for SNMP version 1. The classes have been designed to be adapted to SNMP Version 2 and some of the V2 capability already exists. Many of the SMI structures for SNMP version 2 are already present in SNMP++.

SNMP Get, Get Next and Set Supported

The SNMP class supports three access methods for getting and setting MIB variables. All three member functions utilize similar parameter lists and operate in a blocked or non-blocked (asynchronous) manner.

Redefinition Through Inheritance

SNMP++ is implemented using C++ and thus allows a programmer to overload or redefine behavior which does not suite their needs [Stroustrup]. For example, if an application requires special Oid object needs, a subclass of the Oid class may be created, inheriting all the attributes and behavior the Oid base class while allowing new behavior and attributes to be added to the derived class.

Many Engine

The many engine provides an easy to use interface for getting or setting objects in bulk using SNMP version 1. Using a single member function call, the caller may retrieve up to fifty objects from the specified agent. For SNMP v1, the many engine breaks up the request into multiple request PDUs based on maximum PDU size. As SNMP++ migrates to SNMP v2, the many engine will utilize v2's *awesome* get-bulk request.

SNMP++ for Windows 3.1

SNMP++ has currently been tested and runs over MS-Windows 3.1, MS-Windows For Work Groups 3.11, MS-Windows NT 3.5, and MS-Windows '95 Beta II. SNMP++ for MS-Windows utilizes WinSNMP for its SNMP Basic Encoding Rules (BER) and transport services. This includes the encoding and decoding of Protocol Data Units (PDUs) and transporting them over WinSockets or Novells NWIPXSPX . SNMP++ relies on a robust and reliable WinSNMP.DLL. The hope and intent is that as WinSNMP solidifies and matures, SNMP++ will run over any WinSnmp.DLL implementation.

Runs Over WinSnmp Ver 1.1

WinSNMP ver 1.1 is required to run SNMP++ on MS-Windows. WinSNMP Version 1.0 will not work since the interface has changed.

Multiple Sessions Via Multiple Instances

WinSNMP supports a session model. Sessions are supported in SNMP++ through different instances of the SNMP class. Each instance creates and maintains its own session. The number of instances allowed is limited only by the WinSNMP.DLL and WinSock.DLL being used. A program may create and use different SNMP objects for different sessions. *Note!, since each session maps to an underlying UDP or IPX socket, you may need to fine-tune your stack to allow more sockets.*

Multiple Concurrent Blocked Mode Requests

Using different SNMP class instances, multiple blocked Snmp::gets or Snmp::sets requests may be evoked concurrently. This feature leans heavily on the robustness of the WinSNMP.DLL. For example, a windows timer may trigger a get request on a given SNMP object. While this request is pending, another timer on a different object may fire which causes a different request to be issued. SNMP++ manages this scenario by: 1) allowing Windows messages to be processed while waiting for a PDU response, and 2) queuing incoming PDU responses in a container class.

IP and IPX Support using FTP Software Inc.'s WinSNMP.DLL

By utilizing FTP's WinSNMP, IP and IPX support are available. For IP operation, a WinSock compliant stack is required. For IPX, a Netware client and the required Netware drivers are needed. SNMP++ has been tested to run over a wide variety of protocol stacks including FTP, Netmanage, LanWorkPlace, MS-WFWG 3.11, and Windows NT.

IP Support Using American Computer and Electronics Corp. Netplus WinSNMP.DLL

Utilizing ACECs NetPlus WinSNMP, IP support is available. A Winsock compliant stack is required. Contact ACEC for supported WinSock stacks.

Windows Message Handling

While blocking on a Snmp::get, Snmp::get_next , or Snmp::set, SNMP++ allows other Windows messages to be processed. Without this feature, the entire Windows application would be tied up while waiting for the response PDU. There are times when an application may want to terminate while in a pending blocked mode request. SNMP++ provides facilities for globally shutting down all pending blocked requests *global shutdown*, and partially shutting down just one SNMP++ session, *partial shutdown*. This allows an application to shut down on the fly and not have to wait for outstanding requests to complete.

Medium or Large Model Support

SNMP++ may be compiled and used in both medium and large memory models.

Rendezvous Shut Down Messages

MS-Windows SNMP++ supports shut down messages for shutting down a blocked SNMP++ request. This allows an application to shut down a request without waiting for it to finish.

Runs on MS-Windows NT

SNMP++ applications for Win16 may run on Windows NT using the native NT Winsock compliant stack. This does not offer the performance of a Win32 application but does allow execution on the NT platform.

Trap Support

SNMP++ includes support for interfacing with WinSNMP trap mechanisms. This includes arming, filtering and receiving traps. The interface for traps utilizes the asynchronous mode of SNMP++.

Compatability with HP OpenView for Windows

A number of applications have been created using SNMP++ which coexist and are compatible with HP's OpenView for MS-Windows. This includes full SNMP support and the reception of traps.

The PDU Container Class

In order to accommodate multiple concurrent blocked mode requests, a container class is used to hold the incoming response PDUs. As an SNMP object is instantiated, a new session and new Windows class is created. This procedure is used to process any incoming PDUs for that session. This hidden window remains for the life of the SNMP object and WinSNMP will call it whenever a PDU has arrived for that session. The windows procedure processes the WinSNMP notification by receiving the PDU, verifying that it is valid, and stuffing it into the PDU container. By default, the container can handle twenty concurrent requests. The verification includes verifying that the response PDU matches a pending request and contains no errors. If it is invalid, the PDU is discarded automatically. In addition to storing incoming PDUs, the container class serves PDUs to the waiting blocked process. While a process is waiting in a message pump loop, it queries the container class for the PDU matching the one it had issued a request for. If the PDU is present in the container before the timeout period, the process then extracts the PDU from the container and uses it. In addition to processing and serving PDUs, the PDU class maintains statistics PDU traffic. These statistics include...

- Number of Received PDUs
- Number of Transmitted PDUs
- Number of Time-outs
- Number of Send Errors
- Number of Receive Errors

The Bottom Line: The programmer does not need to know anything about the PDU Container Class or its mechanisms unless you are interested in obtaining performance statistics.

Tested Over MFC and 3.1 API

SNMP++ / MS-Windows applications have been created and tested using Microsoft's Visual C++ Foundation Classes (MFC) and using the standard 3.1 API.

SNMP++ for HPUX

Runs Using SNMP Research's SNMP Libraries

The HPUX implementation offers the identical interface and behavior as SNMP++ for MS-Windows. SNMP++ for HPUX is compatible with HP OpenView for HPUX. A number of SNMP++ applications have been created for HP OpenView for MS-Windows and then ported to HP OpenView for HPUX using SNMP++ as the portable SNMP interface.

Identical Class Interface

The class interface for the UNIX implementation is identical to MS-Windows. Only the internal class implementation of the SNMP class have been changed.

Portable to UNIX-Windows Emulators

SNMP++ runs over HPUX by compiling and linking the proper SNMP++ class implementation. SNMP++ / HPUX is designed to run in a native text mode HPUX app, in a X-Window app, or using Windows-to-UNIX porting tools.

Multiple Connections via Multiple Instances

Under HPUX, the concept of multiple SNMP++ object instances mapping to unique UDP connections has been preserved. Each SNMP++ object maintains and manages its own UDP socket. This ensures that Windows code will have the same behavior in a HPUX environment.

Note! Async mode and traps are not yet available for SNMP++ on HPUX.

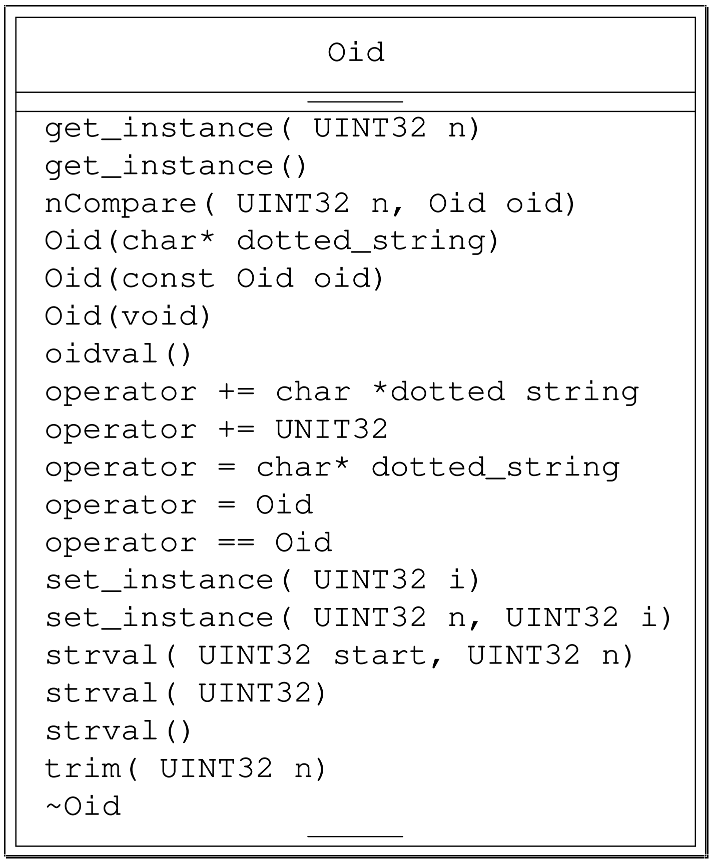
The Object Identification Class

The Object Identification (Oid) class is the encapsulation of an SNMP object identifier. The SMI Oid, its related structures and functions, are a natural fit for object orientation. In fact, the Oid class shares many common features to the C++ String class. For those of you familiar with the C++ String class or MFC's Cstring class, the Oid class will be familiar and easy to use. The Oid class is designed to be efficient and fast. Do not make the false assumption that by using C++, a performance penalty must be paid. A well encapsulated C++ class is easier to fine tune than the equivalent C code [Meyers]. The Oid class allows definition and manipulation of object identifiers. The Oid Class is fully portable and does not rely on WinSNMP or any other Windows or UNIX SNMP API to be present. The Oid class may be compiled and used with any ANSI C++ compiler. The Oid class includes all the related SMI types for Oids.

Object Modeling Technique Representation

The Object Modeling Technique (OMT) methodology was used to design all SNMP++ classes . OMT is a popular design methodology for modeling objects [Rumbaugh].

OMT Public View Of Oid Class



```
Oid

get_instance( UINT32 n)
get_instance()
nCompare( UINT32 n, Oid oid)
Oid(char* dotted_string)
Oid(const Oid oid)
Oid(void)
oidval()
operator += char *dotted string
operator += UNIT32
operator = char* dotted_string
operator = Oid
operator == Oid
set_instance( UINT32 i)
set_instance( UINT32 n, UINT32 i)
strval( UINT32 start, UINT32 n)
strval( UINT32)
strval()
trim( UINT32 n)
~Oid
```

Oid Class Public Member Functions

There are a variety of public member functions which allow for the construction, destruction, access and modification of Oid objects.

Oid Class Constructors & Destructors

Oid objects may be instantiated either statically or dynamically depending on your need.

An Oid object may be constructed with no arguments. Using this constructor, the Oid object has no value, but can be mutated to change its value.
// constructor using no arguments
Oid::Oid(void);

An Oid object may be constructed with a character array representing the object id.
// constructor using a dotted string
*Oid::Oid(const char WINFAR * dotted_oid_string);*

An Oid object may be constructed using another Oid object.
// constructor using another oid object
Oid::Oid (const Oid &oid);

The destructor for the Oid object frees up any memory occupied by the object.
// destructor
Oid::~Oid();

Oid Class Overloaded Operators

Various operators are overloaded which allow comparison and mutation of Oid objects. Overloaded operators allow easy assignment and comparison of Oid objects.

Assign an Oid object a dotted string value.
// assignment to a string operator overloaded
*Oid::Oid& operator=(const char WINFAR *dotted_oid_string);*

Assign an Oid object another Oid Object
// assignment to another oid object overloaded
Oid::Oid& operator=(const Oid &oid);

Test the equivalence of two Oid objects
// equivalence operator overloaded
friend int operator==(Oid &x,Oid &y)

Append a dotted string to an existing Oid object
// append operator, appends a string
*Oid::Oid& operator+=(const char WINFAR *a);*

Append a unsigned long int as the last instance of an Oid object
// appends an int
Oid::Oid& operator+=(const unsigned long i);

Oid Class String Value Methods

Oid class string value member functions allow retrieval of an Oid objects dotted string representation. This is valuable when printing out an Oid object.

Return the entire dotted string representation
// return an oid as a dotted string value
*char * Oid::strval();*
Return the dotted string representation where n specifies the position from the rightmost value.
// return dotted string value from the right
// where the user specifies how many positions to print
*char * Oid::strval(unsigned long n);*
Return the dotted string representation where start specifies the beginning location and n specifies how many values to right.
// return a dotted string where the caller specifies
// where the starting position is and how many to include to the right
*char * Oid::strval(unsigned long start, unsigned long n);*

Oid Class Set Instance & Get Instance Methods

The set and get instance member functions allow setting and getting individual Oid object values. These methods are particularly useful when intricate Oid object manipulation is required, such as when implementing get-nexts.

The set instance method allows changing the rightmost instance of an Oid object.

```
// set the rightmost instance
void Oid::set_instance( unsigned long i)
```

The set instance with an argument allows modifying an Oid object at position n.

```
// modify position n of an oid to value i
// indexes are 1 to n
void Oid::set_instance(unsigned long  n,     // instance # to change
                              unsigned long i)    // new value
```

The get instance member function allows retrieval of the rightmost Oid object value.

```
// returns the rightmost value of the oid
unsigned long Oid::get_instance()
```

The get instance with an argument allows retrieval of an Oid object value at position n.

```
// returns the value of an oid
// at position n
unsigned long Oid::get_instance(unsigned long  n)
```

Oid Class Trim Method

The trim member function allows trimming off the n rightmost values of an Oid object.

```
// trim off the n rightmost values of an oid
void Oid::trim(unsigned long n)
```

Oid Class nCompare Method

The nCompare method allows comparing two Oid objects where n specifies the n leftmost values of each Oid object to compare.

```
// compare the n leftmost bytes
// returns TRUE or FALSE
int Oid::nCompare(unsigned long n, const Oid &o);
```

Oid Class Examples

The following examples show different ways in which to use the Oid class. The Oid class does not require or depend on any other libraries or modules. The following code is ANSI C++ compatible.

```
#include “oid.h”
void oid_example_1()
{
   // construct an Oid with a dotted string and print it out
   Oid o1("1.2.3.4.5.6.7.8.9.1");
   printf(“o1=%s”,o1.strval());

   // construct an Oid with another Oid and print it out
   Oid o2(o1);
   printf(“o2=%s”,o2.strval());

   // trim o2’s last value and print it out
   o2.trim(1);
   printf(“o2=%s”,o2.strval());

   // add a 2 value to the end of o2 and print it out
   o2+=2;
   printf(“o2=%s”,o2.strval());

   // create a new Oid, o3
   Oid o3;

   // assign o3 a value and print it out
   o3="1.2.3.4.5.6.7.8.9.3";
   printf(“o3=%s”,o3.strval());

   // create o4
   Oid o4;

   // assign o4 o1’s value
  o4=o1;

   // trim off o4 by 1
   o4.trim(1);

   // concat a 4 onto o4 and print it out
   o4+=4;
   printf(“o4=%s”,o4.strval());

   // make o5 from o1 and print it out
   Oid o5(o1);
   printf(“o5=%s”,o5.strval());
```

Oid Example Continued...

```
  // compare two not equal oids
  if (o1==o2)   printf("O1 EQUALS O2");
  else   printf("O1 NOT EQUALS O2");

  char msg[100];
  // print out a piece of o1
  sprintf(msg,"strval(3) of O1 = %s", o1.strval(3));
  printf("%s",msg, strlen( msg));

  // print out a piece of o1
  sprintf(msg,"strval(1,3) of O1 = %s", o1.strval(1,3));
  printf("%s",msg, strlen( msg));

  // set o1's last instance
  o1.set_instance(49);
  sprintf(msg,"O1 modified = %s", o1.strval());
  printf("%s",msg, strlen( msg));

  // set o1's last instance
  o1.set_instance(3,49);
  sprintf(msg,"O1 modified = %s", o1.strval());
  printf("%s",msg, strlen( msg));

  // get the last instance of 02
  sprintf(msg,"last of o2 = %ld",o2.get_instance());
  printf("%s",msg,strlen(msg));

  // get the 3rd instance of 02
  sprintf(msg,"3rd of o2 = %ld",o2.get_instance(3));
  printf("%s",msg,strlen(msg));

  // ncompare
  if (o1.nCompare(3,o2))
     printf("nCompare o1,o2,3 ==");
  else
    printf("nCompare o1,o2,3 !=");

  // make an array of oids
  Oid oids[30]; int w;
  for ( w=0;w<30;w++)
  {
   oids[w] = "300.301.302.303.304.305.306.307";
    oids[w] += (w+1);
  }
  for (w=0;w<25;w++)
  {
    sprintf( msg,"Oids[%d] = %s",w, oids[w].strval());
    printf("%s",msg, strlen(msg));
    }
}
```

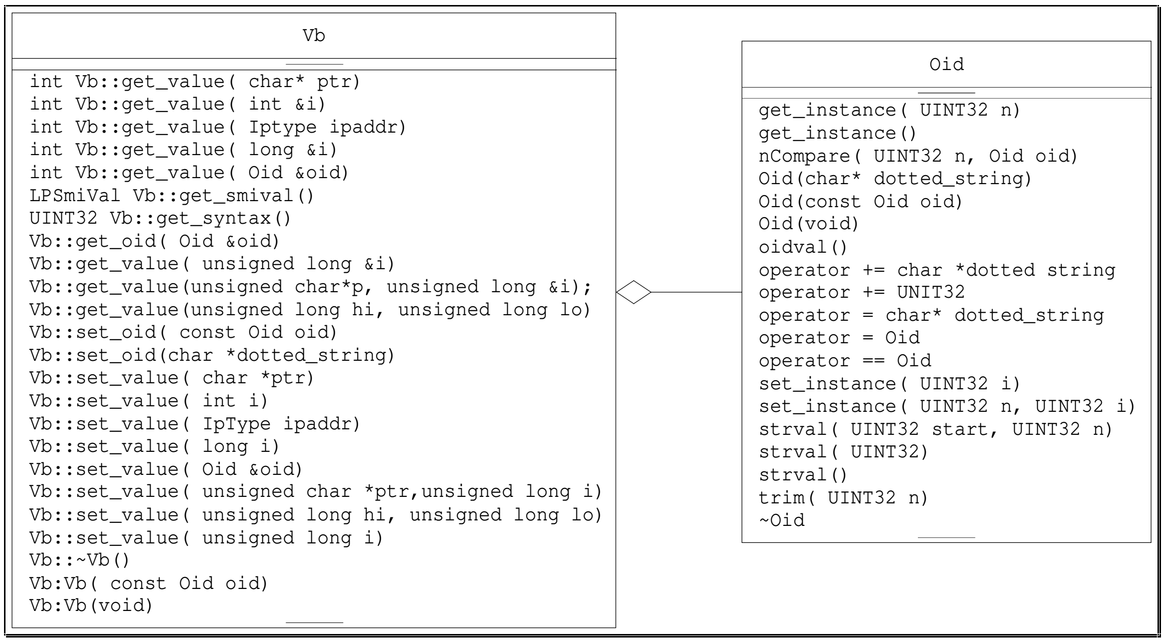
The Variable Binding Class

The variable binding (Vb) class represents the encapsulation of a SNMP variable binding. A variable binding is the association of a SNMP object ID with its SMI value. In object oriented methodology, this is simply a *has a* relation. A Vb object *has an* Oid object and a SMI value. The Vb class allows the application programmer to instantiate Vb objects and assign the Oid portion (Vb::set_oid), and assign the value portion (Vb::set_value). Conversely, the Oid and value portions may be extracted using Vb::get_oid() and Vb::get_value(). The public member functions Vb::set_value() and Vb::get_value() are overloaded to provide the ability to set or get different values to the Vb binding. Variable binding lists in SNMP++ are represented as arrays of Vb objects. All SMI types are provided within the Vb Class. The Vb class provides full data hiding. The user does not need to know about SMI value types, Oid representations, or other related SNMP structures. Like the Oid class, the Vb class is fully portable using a standard ANSI C++ compiler.

Object Modeling Technique Representation

The Object Modeling Technique (OMT) was used to design the Variable Binding (Vb) Class. A Vb object is related to Oid object since every Vb object *has an* Oid object. This is a *one-to-one* association.

OMT Public View of Vb Class


Vb
int Vb::get_value(char* ptr)
int Vb::get_value(int &i)
int Vb::get_value(Iptype ipaddr)
int Vb::get_value(long &i)
int Vb::get_value(Oid &oid)
LPSmiVal Vb::get_smival()
UINT32 Vb::get_syntax()
Vb::get_oid(Oid &oid)
Vb::get_value(unsigned long &i)
Vb::get_value(unsigned char*p, unsigned long &i);
Vb::get_value(unsigned long hi, unsigned long lo)
Vb::set_oid(const Oid oid)
Vb::set_oid(char *dotted_string)
Vb::set_value(char *ptr)
Vb::set_value(int i)
Vb::set_value(IpType ipaddr)
Vb::set_value(long i)
Vb::set_value(Oid &oid)
Vb::set_value(unsigned char *ptr,unsigned long i)
Vb::set_value(unsigned long hi, unsigned long lo)
Vb::set_value(unsigned long i)
Vb::~Vb()
Vb:Vb(const Oid oid)
Vb:Vb(void)
Oid
get_instance(UINT32 n)
get_instance()
nCompare(UINT32 n, Oid oid)
Oid(char* dotted_string)
Oid(const Oid oid)
Oid(void)
oidval()
operator += char *dotted string
operator += UNIT32
operator = char* dotted_string
operator = Oid
operator == Oid
set_instance(UINT32 i)
set_instance(UINT32 n, UINT32 i)
strval(UINT32 start, UINT32 n)
strval(UINT32)
strval()
trim(UINT32 n)
~Oid

Vb Class Public Member Functions

The Vb class provides a variety of public member methods to access and modify Vb objects. The Vb class requires presence and use of the Oid class.

Vb Class Constructors & Destructors

A Vb object may be constructed with no arguments. In this case, the Oid and value portions must be set with subsequent member function calls.

```
// constructor with no arguments
// makes an vb, un-initialized
Vb::Vb( void);
```

Alternatively, a Vb object may be constructed with an Oid object as a construction parameter. This initializes the Oid part of the Vb object to the Oid passed in. The Vb object makes a copy of the Oid passed in. This saves the programmer from having to worry about the duration of the parameter Oid.

```
// constructor to initialize the oid
// makes a vb with oid portion initialized
Vb::Vb( const Oid oid);
```

The destructor for a Vb object releases any memory and/or resources which were occupied. For statically defined objects, the destructor is called automatically when the object goes out of scope. Dynamically instantiated objects require usage of the *delete* construct to cause destruction.

```
// destructor
// if the vb has a oid or an octect string then
// the associated memory needs to be freed
Vb::~Vb();
```

Vb Class Get Oid / Set Oid Member Functions

The get and set Oid member functions allow getting or setting the Oid part of a Vb object. When doing SNMP gets or sets, the variable is identified by setting the Oid value of the Vb via the Vb::set_oid(Oid oid). Conversely, the oid portion may be extracted via the Vb::get_oid(Oid &oid) member function. The get_oid member function is particularly useful when doing SNMP get_next's.

The Oid portion of a Vb object can be set with an already constructed Oid object

```
// set value oid only with another oid
void Vb::set_oid( const Oid &oid);
```

Alternatively, the Oid portion may be set with the dotted string representation of the Oid.

```
// set oid value with a const string
void Vb::set_oid( const char WINFAR * dotted_oid_string);
```

The Oid portion may be retrieved by providing a target Oid object. This destroys the previous value of the Oid parameter object.

```
// get oid portion
void Vb::get_oid( Oid &oid);
```

Vb Class Get Value / Set Value Member Functions

The get_value, set_value member functions allow getting or setting the value portion of a Vb object. These member functions are overloaded to provide getting or setting different types. The internal hidden mechanisms of getting or setting Vb's handles all memory allocation/de-allocation. This frees the programmer from having to worry about SMI-value structures and their management. Get value member functions are typically used to get the value of a Vb object after having done a SNMP get. Set value member functions are useful when wishing to set values of Vb's when doing a SNMP set. The get_value member functions return a -1 if the get does not match what the Vb is holding.

Set the value portion of a Vb object to an integer.

```
// set the value with an int
void Vb::set_value( int I);
```

Set the value portion of a Vb Object to a long integer.

```
// set the value with a long signed int
void Vb::set_value( long int I); unsigned long
```

Set the value portion of a Vb object to an unsigned long integer.

```
// set the value with an unsigned long int
void Vb::set_value( unsigned long int i);
```

Set the value portion of a Vb object to two unsigned ints. This is used for SNMP 64 bit counters comprised of a hi and lo 32 bit portion.

```
// set value for building an 64 bit counter
void Vb::set_value( unsigned long int hi,unsigned long int lo);
```

Set the value portion of a Vb object to an Oid.

```
// set value for setting an oid
// creates own space for an oid which
// needs to be freed when destroyed
void Vb::set_value( Oid &varoid);
```

Set the value portion of a Vb object to an octet string. This is comprised of a unsigned char string and a length.

```
//set value for setting an octet string
// creates own space for a octet which
// needs to be freed when destroyed
void Vb::set_value( unsigned char WINFAR * ptr, unsigned long len);
```

Set the value portion of a Vb object to a char string. Really, this internally uses the SMI value portion of an octet string but makes it easier to use when it is an ASCII string. (eg system descriptor)

```
// set value on a string
// makes the string an octet
// this must be a null terminates string
void Vb::set_value( char WINFAR * ptr);
```

Set the value portion of a Vb to an IP address. The typedef for Iptype is defined in Vb.hpp and is simply a 4 byte octet string. IP address is a explicit SMI value type.

```
// set an ip address as a value
void Vb::set_value ( Iptype ipaddr);
```

Vb Class Get Value Member Functions

All Vb::get_value member functions modify the parameter passed in. If a Vb object does not contain the requested parameter type, the parameter will not be modified and a -1 will be returned. Otherwise on success, a 0 status is returned.

Get an integer value from a Vb object.

```
// get value int
// returns 0 on success and value
int Vb::get_value( int &i);
```

Get a long integer from a Vb object.

```
// get the signed long int
// returns 0 on success and a value
int Vb::get_value( long int &I);
```

Get an unsigned long integer value from a Vb.

```
// get the unsigned long int
// returns 0 on success and a value
int Vb::get_value( unsigned long int &i);
```

Get two unsigned longs from a Vb object. (64 counter hi,lo).

```
// get value for building an 64 bit counter
// returns 0 on success and a value
int Vb::get_value( unsigned long int &hi,unsigned long int &lo);
```

Get an Oid object from a Vb object.

```
// get the oid value
// free the existing oid value
// copy in the new oid value
int Vb::get_value( Oid &varoid);
```

Get an unsinged char string value from a Vb object (Octet string).

```
// get a unsigned char string value
// destructive, copies into given ptr of up
// to len length
//
// Note! the caller must provide a target string big
// enough to handle the vb string, else memory corruption
int Vb::get_value( unsigned char WINFAR * ptr, unsigned long &len);
```

Get a char string from a Vb object. This grabs the octet string portion and pads it with a null.

```
// get a char * from an octet string
// the user must provide space or
// memory will be stepped on
int Vb::get_value( char WINFAR *ptr);
```

Get a IP address from a Vb object. Iptype is defined as an array of four unsigned chars.

```
// get an ip address
int Vb::get_value( Iptype ipaddr);
```

Vb Object Get Syntax Member Function

This method violates the object oriented paradigm. An object knows what it is. By having a method which returns the id of an object violates its data hiding. Putting that aside, there are times when it may be necessary to know what value a Vb is holding to allow extracting that value. For example, when implementing a browser it would be necessary to grab a Vb, ask it what it has and then pull out whatever it may hold.

```
// return the current syntax
// This method violates the OO paradigm but may be useful if
// the caller has a vb object and does not know what it is.
// This would be useful in the implementation of a browser.
unsigned long get_syntax();
```

Vb Class Examples

The following examples show different ways in which to use the Vb class. The Vb class does not require or depend on any other libraries or modules other than the Oid class. The following C++ code is ANSI compatible.

```
#include “oid.h”
#include “vb.h”
vb_test()
{
   // -------[Ways to construct vb objects ]-------
   // construct a single vb object
   Vb vb1;

   // construct a vb object with an Oid object
   // this sets the oid portion of the vb
   Oid d1(“1.3.6.1.4.12”);
   Vb vb2(d1);

   // construct a vb object with a dotted string
   Vb vb3(“1.2.3.4.5.6”);

   // construct an array of ten vb’s
   Vb vbs[10];

   //------[Ways to set and get the oid portion of Vb objects ]

   // set and get the oid portion
   Oid d2(“1.2.3.4.5.6”);
   vb1.set_oid(d2);
   Oid d3;
   vb1.get_oid(d3);
   if (d2==d3) printf(“They better be equal!!\n”);

   Vb ten_vbs[10];
   int z;
   for (z=0;z<10;z++)
   ten_vbs[0].set_oid(“1.2.3.4.5”);

   //-------[ ways to set and get values ]

   // set & get ints
   int x,y;
   x=5;
   vb1.set_value(x);
   vb1.get_value(y);
   if ( x == y) printf(“x equals y\n”);
   // set and get long ints
   long int a,b;
   a=100;
```

Vb Class Example Continued..

```
//-------[ ways to set and get values ]
  // set & get ints
  int x,y;
  x=5;
  vb1.set_value(x);
  vb1.get_value(y);
  if ( x == y) printf("x equals y\n");
  // set and get long ints
  long int a,b;
  a=100;
  vb2.set_value( a);
  vb2.get_value(b);
  if ( a == b) printf("a equals b\n");
  // set & get unsigned long ints
 unsigned long int c,d;
 c = 1000;

  vbs[0].set_value( c);   vbs[0].get_value( d);
  if ( c == d) printf("c equals d\n");

  // get and set a 64 bit counter
  unsigned long int  hi,lo;
  unsigned long int big,small;
  hi = 1000;   lo = 1001;
  vbs[1].set_value( hi,lo);
  vbs[1].get_value( big, small);
  if ( (hi==big) && ( lo == small)) printf( "hi == big and lo == small\n");

  // get and set an oid as a value
  Oid o1, o2;
  o1 = "1.2.3.4.5.6";
  vbs[2].set_value( o1);   vbs[2].get_value( o2);
  if ( o1 == o2) printf("o1 equals o2\n");

  // set and get an octet string
  unsigned char data[4],outdata[4];
  unsigned long len,outlen;
  len =4;  data[0] = 10; data[1] = 12; data[2] = 12; data[3] = 13;
  vbs[3].set_value( data,len);
  vbs[3].get_value( outdata, outlen);

  // get & set a string
  char beer[20];   char good_beer[20];
  strcpy( beer,"Sierra Nevada Pale Ale");
  vbs[4].set_value( beer);
  vbs[4].get_value( good_beer);
  printf("Good Beer = %s\n",good_beer);
```

Vb Class Example Continued..

```
   // get and set an ip an address
   IpType ipaddr,new_ipaddr;
   ipaddr[0] = 10;
   ipaddr[1] = 4;
   ipaddr[2] = 8;
   ipaddr[3] = 69;
   vbs[5].set_value( ipaddr);
   vbs[5].get_value( new_ipaddr);
   printf("%d .%d .%d . %d\n", new_ipaddr[0], new_ipaddr[1],
                        new_ipaddr[2],new_ipaddr[3]);

}  // end vb test
```

TimeTicks, Counter and Gauge Classes

These three classes allow the programmer to access or modify SMI timetick, counter and gauge variables. The SMI values are distinguished as separate data types. For all practical purposes, the SNMP++ Timeticks, Counter and Gauge objects can be thought of as unsigned long ints in C++. That is, anything that can be done with an unsigned long int can be done with a TimeTicks, Counter or Gauge object.

TimeTicks Class Example

```
TimeTicks tt;  // declare a time ticks object

vb.get_value( tt);  // extract time ticks from a vb
printf("Time up = %ld", tt);
```

Counter Class Example

```
Counter ctr; // declare a counter object

ctr = 1200;  // assign counter a value
vb.set_value( ctr);  // set a vb with the counter object
```

Gauge Class Example

```
Gauge g; // declare a gauge object

vb.get_value( g);  // get a gauge object from a vb object
printf( "Gauge value = %ld", g);
```

The SNMP Class

The most important class in SNMP++ is the SNMP class. The SNMP class is an encapsulation of a SNMP session. A SNMP session includes a logical connection from an SNMP management station to a managed agent or agents. Handled by the session is the construction, delivery and reception of PDUs. Most APIs require the programmer to directly manage the session. This includes providing a reliable transport mechanism handling time-outs, retries and packet duplication. The SNMP class manages a large part of the session and frees the implementor to concentrate on the agent management. By going through the SNMP class for session management, the implementor is driving through well developed and tested code. The alternative is to design, implement and test your own SNMP engine. The SNMP class manages a session by 1) managing the transport layer over a UDP or IPX connection. 2) handles packaging and un-packaging of Vbs into PDUs 3) provides for delivery and reception of PDUs and 4) manages all necessary SNMP resources.

The SNMP class is easy to use. Three basic methods, Snmp::get, Snmp::set and Snmp::get_next provide the basic functions for a network management application. Blocking or non-blocking access may be used. Multiple sessions may be used each asynchronously firing simultaneous requests. Each session maps to underlying socket so the number of SNMP objects is limited by socket availability. Non-blocking or asynchronous mode requires the programmer to handle request time-outs and retries.

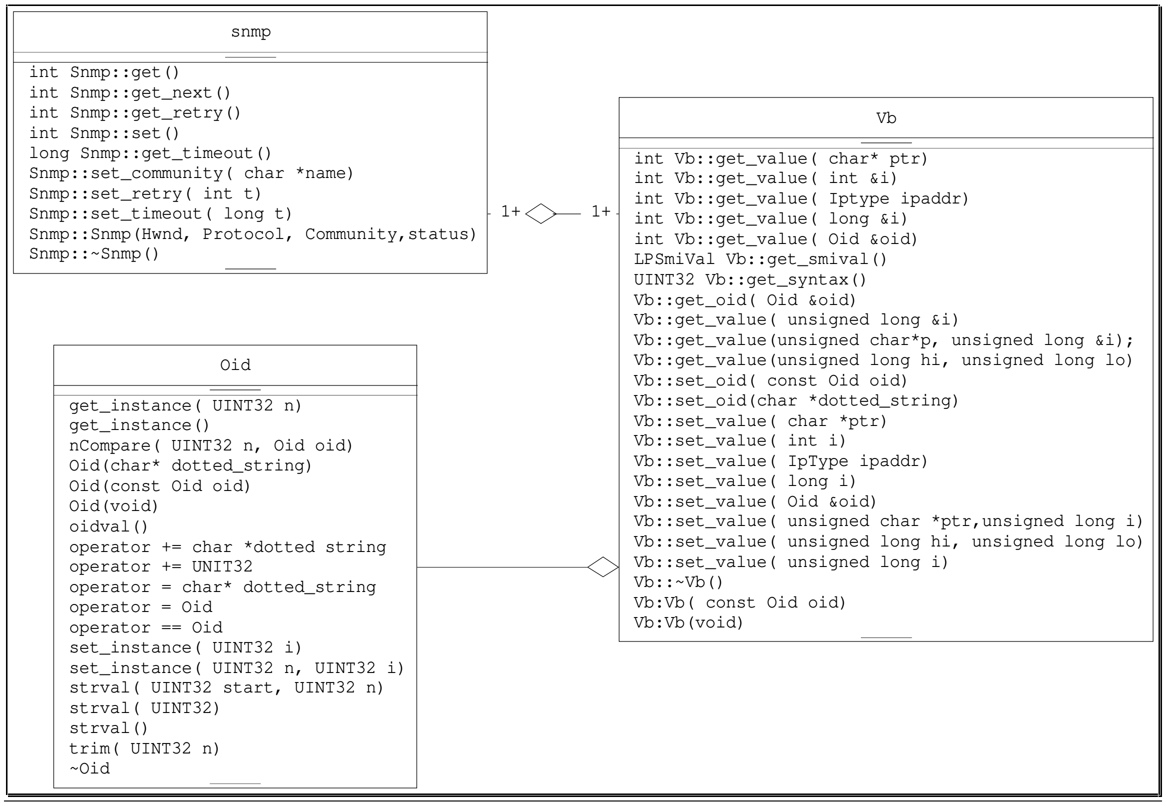
The SNMP class is safe to use. The constructor and destructors allocate and de-allocate all resources needed. This minimizes the likelihood of corrupt or leaked memory. All of the internal SNMP mechanisms are hidden and thus cannot be inadvertently modified.

The SNMP class is portable. The SNMP class interface is portable across OS's and NOS's. The Oid and Vb classes can be compiled and used on any ANSI C++ compiler. The implementation of the SNMP class is platform specific. That is, the SNMP.CPP file is implemented for each OS /NOS platform. Currently this module has been ported to run on MS-Windows over WinSNMP and on HPUX using SNMP++s UNIX engine. The amount of coding needed to port SNMP++ to another platform is minimal. To the application programmer, no code changes are required to move from one platform to the next. *The vast majority of SNMP++ is re-used across platforms; thus, development time and testing time are cut drastically.*

Object Modeling Technique Representation

The SNMP class was designed and developed using the Object Modeling Technique (OMT). Note the relationship between the classes. A SNMP object *has one or more* Vb objects. The Vb objects each *have exactly one* Oid object.

Public View of SNMP Class



```
int Snmp::get()
int Snmp::get_next()
int Snmp::get_retry()
int Snmp::set()
long Snmp::get_timeout()
Snmp::set_community( char *name)
Snmp::set_retry( int t)
Snmp::set_timeout( long t)
Snmp::Snmp(Hwnd, Protocol, Community,status)
Snmp::~Snmp()
```

```
int Vb::get_value( char* ptr)
int Vb::get_value( int &i)
int Vb::get_value( Iptype ipaddr)
int Vb::get_value( long &i)
int Vb::get_value( Oid &oid)
LPSmiVal Vb::get_smival()
UINT32 Vb::get_syntax()
Vb::get_oid( Oid &oid)
Vb::get_value( unsigned long &i)
Vb::get_value(unsigned char*p, unsigned long &i);
Vb::get_value(unsigned long hi, unsigned long lo)
Vb::set_oid( const Oid oid)
Vb::set_oid(char *dotted_string)
Vb::set_value( char *ptr)
Vb::set_value( int i)
Vb::set_value( IpType ipaddr)
Vb::set_value( long i)
Vb::set_value( Oid &oid)
Vb::set_value( unsigned char *ptr,unsigned long i)
Vb::set_value( unsigned long hi, unsigned long lo)
Vb::set_value( unsigned long i)
Vb::~Vb()
Vb:Vb( const Oid oid)
Vb:Vb(void)
```

```
get_instance( UINT32 n)
get_instance()
nCompare( UINT32 n, Oid oid)
Oid(char* dotted_string)
Oid(const Oid oid)
Oid(void)
oidval()
operator += char *dotted string
operator += UINT32
operator = char* dotted_string
operator = Oid
operator == Oid
set_instance( UINT32 i)
set_instance( UINT32 n, UINT32 i)
strval( UINT32 start, UINT32 n)
strval( UINT32)
strval()
trim( UINT32 n)
~Oid
```

SNMP Class Public Member Functions

The SNMP class provides a variety of member functions for creating, managing and terminating a session. Multiple SNMP objects may be instantiated at the same time.

SNMP Class Constructors and Destructors

The constructors and destructors for the SNMP class allow sessions to be opened and closed. By constructing a SNMP object, a SNMP session is open. UDP or IPX sockets are created and managed until the objects are destroyed. SNMP objects may be instantiated dynamically or statically.

SNMP Class Constructor, Blocked Mode

The construction parameters include a window handle, protocol type, community name and a return status. Since constructors do not return values in C++, the caller must provide a status which should be checked after instantiating the object. The window parameter is applicable when programming in Windows only and may be null in environments such as UNIX. The protocol type specifies the type of transport services to use. The community name parameter allows the caller to specify the community name. The community name may be modified at some later time via Snmp::set_community(). The caller should check the return status for 'SNMP_CLASS_SUCCESS'. If the construction status does not indicate success, the session should not be used.

```
// constructor, blocked SNMP object
  Snmp::Snmp( WORD hwnd,               // parent handle
       Protocol protocol,              // type of protocol to use
       const char *community_name,     // community name
       int *status);                   // construction status
```

SNMP Class Constructor, Asynchronous Mode

In order to use an SNMP++ asynchronous member functions, an asynchronous object must be instantiated. Blocking and asynchronous SNMP objects are mutually exclusive. An asynchronous object may not use blocking calls and a blocked object may not use asynchronous calls. One additional parameter is required for async operation. The callback function pointer allows the programmer to specify a function to be called when asynchronous event occurs on the object. This gives the programmer the flexibility to process the async event in any manner desired.

```
 // constructor SNMP async object
Snmp( WORD hwnd,                      // parent handle
      Protocol protocol,              // type of protocol to use
      const char *community_name,     // community name
      snmp_callback lpcallback,       // callback function address
      int *status);                   // construction status
```

The snmp_callback typedef is the function prototype for the call back. When the callback is called, the Vb objects contain the payload for the response PDU. Error status, error index, address and community name information are provided to allow differentiating on PDU from another.

```
typedef void (WINFAR *snmp_callback)(long int,          // request id
                            Vb*,                        // vbs
                            int,                        // number of vbs
                            long int,                   // error status
                            long int,                   // error index
                            unsigned char *agent_addr,  // agent address
                            unsigned char *community,   // community name
                            unsigned long community_len); // len of comm. name
```

SNMP Class Destructor

The SNMP class destructor closes the session and releases all resources and memory.

```
// destructor
   Snmp::~Snmp();
```

SNMP Class Access and Mutator Member Functions

SNMP Class Set Timeout & Get Timeout

By default, when an Snmp object is instantiated, automatic time-outs and retries are set to one second and one retry respectively. Blocked request member functions, (get set and get_next) will utilize the automatic timeout and retry parameters. Timeout values may be accessed and modified using the Snmp::set_timeout and Snmp::get_timeout member functions.

Set the timeout value. Parameter is the number of milliseconds to wait.

```
// set timeout
 void Snmp::set_timeout( DWORD t)
```

Get timeout allows the timeout value to be accessed. The returned value in milliseconds.

```
// get timeout
DWORD Ssnmp:: get_timeout()
```

Snmp Class Set Retry & Get Retry

The set and get retry member functions allow accessing and modifying the retry behavior of the SNMP class request member functions (get, set and get_next). By default, the retry value is set to one when an Snmp object in instantiated.

Set the retry value.

```
// set retry
 void Snmp::set_retry( int r )
```

Get the retry value.

```
// get retry
 int Snmp::get_retry( )
```

SNMP Class Set Community Name

The set community name member function allows modification to the community name which will be used when requesting data through the request member functions. The initial value of the community name is passed in as a construction parameter.

```
// set the community name
void Snmp:: set_community( const char * new_name);
```

Or community names may be set using a pointer and a len.

```
// set community name using ptr and len
void Snmp::set_community( const char * new_name, int len);
```

SNMP Class Request Member Functions

In order to access or modify an agents MIB, requests must be made via the Snmp::get, Snmp::set or Snmp::get_next. All of these member functions accept the identical parameter lists. All blocked mode requests behave the same way in terms of timeout and retries and all return the same success and error status values.

Request Member Function Parameter Description

Blocked mode request member functions, require five parameters and return an error status. The first argument is a pointer to an array of Vb objects. The second parameter specifies the size of the array. The third and fourth parameters map directly to the received Pdu's error status and error index. The last parameter specifies the destination address. Asynchronous calls require only four parameters.

- **Parameter Vb***

The first parameter of the request methods specify a pointer to an array of Vb objects. Variable binding lists in SNMP++ are represented as arrays of Vb objects. These objects are the variable bindings to be applied in the set or get. The caller may specify any number of Vb's as long as the max PDU size is not overrun. An error status will be returned if this is the case.

- **Parameter vb_count**

The second parameter specifies the number of Vb objects passed in the first parameter.

- **Parameter long int err_status**

This parameter is the error status of the received response PDU. If this value is non zero, the return status of the request member function will flag it as 'SNMP_ERR_STATUS_SET'. This parameter is valuable when accessing objects incorrectly or accessing objects which do not exist. This return parameter is the SMI error status and may have the following values..

0 - No error
1 - PDU too Big
2 - No such MIB var name
3 - Bad MIB var value
4 - Read Only MIB var
5 - General Error

- **Parameter err_index**

This parameter specifies the Vb object in error. Note, the error index is one not zero based. The first Vb will be flagged as index number 1.

- **Parameter dest_addr**

  The last parameter specifies the destination address where the request shall be directed. This address may in the form of an IP or IPX address depending on the type of transport services available.

SNMP Class Blocked Get Member Function

The get member function allows getting objects from the agent at the specified address. In order to use the get, the user needs to fill the Vb objects with the requested Oid's. Returned will be the Vb's with their values set. Blocked member functions will return as soon as the SNMP response is received or if a timeout or error condition has occurred.

```
//--------[ get ]------------------------------------------
int    get( Vb *vb,                    // pointer to array of vb objects
            int vb_count,              // count of vb objects
            long int &err_status,      // returned error status
            long int &err_index,       // returned error index
            const char *dest_addr)     // get address
```

SNMP Class Blocked Get Next Member Function

The get next member function may be used to traverse an agents MIB. Get next may traverse more than one table at a time. (In most cases just using one Vb will be enough). The caller fills the request Vb with the Oid requested. Returned will be the Oid and the value for the next table entry. The caller may then call get_next again with the returned Vb. The caller must determine when to stop calling get_next based on return Vb Oid values.

```
//---------[ get next ]-------------------------------------
  int get_next(Vb *vb,                 // pointer to array of vb objects
            int vb_count,              // count of vb objects
            long int &err_status,      // returned error status
            long int &err_index,       // returned error index
            const char *dest_addr)     // get address
```

SNMP Class Blocked Set Member Function

The set member function allows setting agent objects. The caller fills in the Vb's to be set with the Oid values. Returned will be the status of the set.

```
//---------[ set ]-------------------------------------------
  int    set( Vb *vb,                // pointer to array of objects
              int vb_count,          // count of objects
              long int err_status,   // returned error status
              long int err_index,    // returned error index
              const char *dest_addr)  // target address
```

SNMP Class Asynchronous Member Functions

When SNMP++ objects are instantiated as async objects, asynchronous member functions may be utilized. For async SNMP++ objects, it is the programmers responsibility to handle time-outs and retries. Async objects lend themselves well to SNMP access which occurs repeatedly. This includes updating graphs or any other time driven event where retries will occur automatically.

SNMP Class Asynchronous Get Member Function

The async get allows getting SNMP objects from the specified agent. The async get call will return as soon as the request PDU has been sent. It does not wait for the response PDU. The programmers defined callback, which was specified upon the SNMP's object async instantiation, will be called when the response PDU has arrived. The implementation of the callback may utilize the response payload in any desired manner.

```
//-----------------------[ get async ]---------------------------------
  int Snmp:: get_async( Vb *vb,              // pointer to array of vb objects
                        int vb_count,        // count of vb objects
                        long int req_id,     // request id to use, returned
                        const char *dest_addr)  // address to send to
```

SNMP Class Asynchronous Set Member Function

The asynchronous set member function works in the same manner as the get counter part.

```
//------------------------[ set async ]---------------------------------
  int Snmp::set_async( Vb *vb,                     // pointer to array of vb objects
                       int vb_count,               // count of vb objects
                       long int req_id,            // request id to use, returned
                       const char *dest_addr)  // address to send to
```

SNMP Class Asynchronous Get Next Member Function

The asynchronous get-next member function works in the same manner as does async get and async set.

```
//------------------------[ get next async ]-----------------------------
  int get_next_async( Vb *vb,           // pointer to array of vb objects
              int vb_count,              // count of vb objects
              long int req_id,           // request id to use, returned
              const char *dest_addr)     // address to send to
```

Medina’s Many Engine Member Functions

Moises Medina, software engineer at HP Roseville Networks division, extended SNMP++ to include member functions for getting and setting SNMP objects in bulk using SNMP v1. The many engine and the supporting member functions are generic and are part of the SNMP base class. The many engine is convenient when getting or setting many objects of the same type. This comes in handy when getting or setting an entire row of objects. The many engine interface allows getting or setting up to fifty objects in one call. The underlying many engine breaks up the request in separate PDU requests, based on max PDU sizes. An example would be getting port status for a forty eight port hub. With a single call, all forty eight Vb objects can be obtained. The many engine would transparently break up the request into three PDU’s and when finished will return the resulting Vb’s. As SNMP++ migrates to SNMP v2, the internals of the many engine will utilize V2’s *get-bulk*. All many engine member functions require a base Oid. The base Oid specifics the starting Oid used in the many operation. The caller also specifies how many objects are to be gathered. The many engine will start at the base Oid and get all objects up the number specified by concatenating instance values. For example, given a base Oid value of ‘1.2.3.0’ and the number of values of 20, the many engine would get or set values ‘1.2.3.1’ through ‘1.2.3.20’.

SNMP Class Get Many

The Snmp::get_many member function may be used to retrieve ints, long ints, IP addresses and octet arrays with a single call.

```
//---------------------------[ get many ,ints ]---------------------------
  int Snmp::get_many( Oid base_oid,        // base oid to use
               int *ret_values,             // returned int values
               int num_values,              // number of values to get
               long int &err_status,        // returned error status
               long int &err_index,         // returned error index
               const char *dest_addr);  // target address
```

SNMP Class Set Many

```
//---------------------------[ set many ,ints ]--------------------------
  int Snmp::set_many( Oid base_oid,            // base oid to use
               int *in_values,        // values to set
               int num_values,          // number of values to set
               long int &err_status,    // returned error status
               long int &err_index,     // returned error index
               const char *dest_addr);  // target address
```

SNMP Class Trap Methods

SNMP++ allows the reception of traps through use of an SNMP++ asynchronous object. Traps are an asynchronous event therefor asynchronous SNMP object must be used. An important consideration when receiving traps is the concept of trap port ownership. For UDP or IPX sockets, a well known trap port is utilized to receive incoming trap PDU's. Agents throwing traps direct their traps to a defined address and port. The manager listens on the well known port for any incoming traps then receives and processes them. Since on a given machine there is only one well know trap port, it must be shared if more than one application is to receive traps on the machine. This is commonly known as the trap server. For WinSNMP, WinSNMP acts as the trap server. It owns and listens on the well known port. MS-Windows applications may then receive their traps from the WinSNMP.DLL. For other platforms such as Novell's NMS or HPUX Open View, the platforms themselves own the trap port. In order to receive traps on a given platform, SNMP++ must interface with the trap server on that platform. Currently, SNMP++ receives traps only from WinSNMP.

SNMP Class Trap Registration Member Function

SNMP++ for async objects allows registration for the reception of traps through the Snmp::trap_register member function. When a registered trap arrives, it will be directed to the specified call back function which was used when instantiating the SNMP async object. Traps are received as SNMP version 2 format traps, even if they were transmitted as version 1 traps. Version 1 traps are translated to version 2 traps as defined in the SNMPv2 coexistence document [RFC 1452]. In the v2 format for traps, the first Vb is the timestamp, the second vb is the trap id and the third through the last are the payload. The time stamp may be extracted from the Vb object as a TimeTicks object. The trap identifier may be extracted as an Oid object. The parameters on the member function specify the managers address, agents address (agent sending the traps), an Oid mask and a flag for turning the traps on or off. The Oid mask allows filtering specific traps based on their id.

```
//---------------------------[ trap register ]-------------------------
  // To be used only for async Snmp objects
  // Note, WinSnmp v1 informs the trap receiver with SnmpV2 type
  // trap messages. In SnmpV2, the trap format consists of N variable
  // bindings where...
  // - the first vb is the time stamp
  // - the second vb identifies the trap
  // - the third through N vb's are the payload
  //
  //
  int Snmp::trap_register( const char * mgr_addr,    // managers source address
                           const char * agent_addr,  // agents address
                           Oid oidmask,              // oid mask or empty for all
                            int on);                 // boolean for on or off
```

SNMP Class Error Return Codes

There are a variety of return codes when using SNMP++. The error codes are common across platforms and may aid the application programmer in finding and detecting error conditions.

SNMP_CLASS_SUCCESS 0
Operation was Successful.

SNMP_CLASS_START_ERR -1
Transport start up has failed. Verify that the network protocol is in place and is working.

SNMP_CLASS_END_ERR -2
Unable to shut down transport services.

SNMP_CLASS_CONSTRUCT_ERR -3
Unable to construct an SNMP object. Verify that there is enough memory available and that UDP or IPX sockets are available.

SNMP_CLASS_VBL_ERR -4
Internal error while creating an SNMP SMI Vbl.

SNMP_CLASS_OID_ERR -5
Internal error while creating an SMI Oid.

SNMP_CLASS_SETVB_ERR -6
Internal error while setting a SMI Vb.

SNMP_CLASS_ENTITY_ERR -7
Internal error while creating an SNMP entity.

SNMP_CLASS_CONTEXT_ERR -8
Internal error while creating an SNMP context.

SNMP_CLASS_PDUCREATE_ERR -9
Internal error while creating an SMI PDU.

SNMP_CLASS_SEND_ERR -10
Error sending request PDU. This error may occur if the destination address does not exists or if the transport services are not working. Try pinging the device to verify the address exists.

SNMP_CLASS_REC_ERR -11
Error while receiving the PDU response.

SNMP_CLASS_PDUGET_ERR -12
An error occurred while getting the Vbl from the PDU.

SNMP_CLASS_BAD_RESPONSE -13
The specified agent responded with a bad response PDU not matching the request type.

SNMP_CLASS_BAD_ID -14
The agents response did not match the request id. These errors cannot occur in blocked mode since bad responses are discarded automatically.

SNMP_CLASS_BADVB_COUNT	-15
The response PDUs id is a match but the Vb count does not match.	
SNMP_CLASS_TIMEOUT	-16
Timed out while waiting for response PDU. The error can happen and should be treated as a benign condition. Try increasing the default retry or timeout values.	
SNMP_CLASS_ENGINE_BUSY	-17
The SNMP engine was busy. This error should not happen unless you are reentering an already pending request. Pending blocked mode requests should not be reentered.	
SNMP_CLASS_CREATE_FAIL	-18
MS-Windows Only, Unable to create a hidden window class.	
SNMP_CLASS_REG_FAIL	-19
MS-Windows Only, Unable to register a SNMP++ window.	
SNMP_CLASS_CONT_FAIL	-20
MS-Windows Only, Unable to create a PDU container.	
SNMP_CLASS_QUEUE_FULL	-21
MS-Windows Only, the PDU container class is full. Default is twenty concurrent outstanding requests.	
SNMP_ERR_STATUS_SET	-22
The SMI error status flag has been set. See error status and error index for details on error.	
SNMP_CLASS_ILLEGAL_MODE	-23
This error will occur is an asynchronous object is attempted to be used for blocked requests or a blocked object used for asynchronous requests.	
SNMP_CLASS_SHUTDOWN	-25
A blocked mode request received a shutdown request while waiting for the response PDU.	
SNMP_CLASS_TRAP_IN_USE	-26
Failed to register for traps. Trap port may already be in use.	
SNMP_CLASS_TRAP_REG_FAIL	-27
Failed to register for the specific trap is requested.	
SNMP_CLASS_PARTIAL_SHUTDOWN	-28
While waiting for a blocked mode request. A partial shutdown message was processed.	

SNMP Class Examples

Following is a set of examples which illustrate the usage of SNMP++.

SNMP++ Example #1, Getting a Bunch of Values in HPUX

```
#include “snmp.h”
void hpux_example()
{
   int status;
   Vb vb[8];
   long int error_status, error_index;
   char name[255];
   unsigned long long_val;

   // start up the transport services
   transport_start_up( NULL, (Protocol) ip);

   // instantiate a snmp object
   Snmp snmp( NULL, ( protocol) ip, “public”,&status);
   if ( status != SNMP_CLASS_SUCCESS)
   {
      printf(“error constructing snmp object\n”);
      return;
   }

   // set retry and timeout
   snmp.set_retry(3);
   snmp.set_timeout( 2000);  // 2 seconds

   // set up the Vb’s required
   Vb[0].set_oid(“1.3.6.1.2.1.1.1.0”);       // system descriptor
   Vb[1].set_oid(“1.3.6.1.2.1.1.3.0”);       // system up time
   Vb[2].set_oid(“1.3.6.1.2.1.1.2.0”);       // system id
   Vb[3].set_oid(“1.3.6.1.2.1.1.4.0”);       // system contact
   Vb[4].set_oid(“1.3.6.1.2.1.1.5.0”);       // system name
   Vb[5].set_oid(“1.3.6.1.2.1.1.6.0”);       // system location
   Vb[6].set_oid(“1.3.6.1.2.1.1.7.0”);       // system services
   Vb[7].set_oid(“1.3.6.1.2.1.2.2.1.1.0”);  // number of network interfaces

   // get the objects
   status = snmp.get( (Vb*) &vb,8,error_status, error_index,”15.29.32.143”);
   if ( status != SNMP_CLASS_SUCCESS)
   {
      printf(“error getting objects = %d\n”,status);
      return;
   }

   // print out the object values
   vb[0].get_value( name);
   printf(“System Descriptor = %s\n”,name);

}
```

Getting a Bunch of Values in HPUX Continued....

```
   vb[1].get_value( long_val);
   printf("System Up Time = %ld\n",long_val);
   Oid soid;
   vb[2].get_value( soid);
   printf("System Object Id = %s\n", soid.strval());
   vb[3].get_value(  name);
   printf("System Contact = %s\n",name);
   vb[4].get_value(  name);
   printf("System Name = %s\n",name);
   vb[5].get_value( name);
   printf("System Location = %s\n",name);
   vb[6].get_value( long_val);
   printf("System Services = %ld\n",long_val);
   vb[7].get_value( long_val);
   printf("Number of Interface = %ld\n",long_val);

  transport_shutdown( (Protocol) ip);
}  // end hpux example
```

SNMP++ Example #2, Setting Values in MS-Windows MFC

```
void CMainFrame::OnGetst()
{
   // this example sets up a network device for a os image download
   int status;
   char msg[80];
   Vb vb[4];
   unsigned char ipaddr[6] = {10,4,8,82,0,69};
   CDC *cdc;
   long int error_status, error_index;

   // DownLoadStatus
   vb[0].set_oid( DOWNLOAD_STATUS_OID );
   vb[0].set_value( CREATE_AND_GO);

   // DownLoadTDomain
   vb[1].set_oid( DOWNLOAD_DOMAIN_OID);
   Oid oid1( UDP_FAMILY_OID);
   vb[1].set_value( oid1);

   // DownLoadTAddress
   vb[2].set_oid( DOWNLOAD_ADDRESS_OID);
   vb[2].set_value( (unsigned char *) ipaddr,6);

   // DownLoadFileName
   vb[3].set_oid( DOWNLOAD_FILENAME_OID );
   vb[3].set_value( (char *) "test36.dat");

   // construct a snmp object
   Snmp snmp( (WORD) GetSafeHwnd(), (Protocol) ip, "public", &status);

   if ( status != SNMP_CLASS_SUCCESS)
      MessageBox("Unable to Create Snmp Object");
   else
   {
     // do the get
     status = snmp.set((Vb *) &vb,
                       4,
                       error_status,
                       error_index,
                       "10.4.8.78");
     if ( status != SNMP_CLASS_SUCCESS)
     {
        sprintf(msg,"Set Failure = %d",status);
        MessageBox(msg);
     }
     else
     {
       cdc = GetDC( );
       cdc->TextOut( 5,5,"Set Successful !!",17);
       ReleaseDC( cdc);
     }
   }
```

Network Transport Mechanisms

SNMP++ is designed to be portable across multiple OS platforms and run across multiple transport mechanisms. SNMP++ has been designed to run across IP and IPX. In order for SNMP++ to run over these different transport layers, two additional function calls are needed.

Transport Start Up

An application wishing to utilize the SNMP class must first call the transport_start_up function before doing any request member functions. This function verifies that the type of transport service requested is present and working. A fail status will be returned is the service is not available. This function takes an instance pointer (not applicable for UNIX) and a protocol type. Currently IP and IPX are supported.

```
//-----------[ transport layer start up ]------------------------
// Starts up transport services for protocol type passed in.
// Every call to this function should be followed by a
// transport_shut_down. This function must be called prior
// to instantiating and using snmp objects.
int transport_start_up( WORD hInst,Protocol protocol);
```

Transport Shut Down

When an application is complete, transport_shut_down should be called. This function shuts down the transport services for the type specified.

```
//-----------[ transport layer shut down]------------------------
// Shuts down transport services based on the protocol type passed
// in. Every call to this function should be preceeded by a call to
// transport_start_up.
int transport_shut_down( Protocol protocol);
```

SNMP++ Proposed New Features

There are a variety of new features and enhancements which may be included to SNMP++. Extensibility may be accomplished either through inheritance and redefinition or via adding new attributes and behavior to the classes. Below are listed possible enhancements ,they are not ordered.

Support for SNMP version 2

Currently SNMP++ only supports version 1. WinSnmp does not support version 2 at the time this document was authored. Full V2 support includes the following areas.

- **Additional SMI Value Types**
  v2 adds a variety of new SMI types including 64-bit counters and Uinteger types. Some of these features are already in SNMP++.
- **Protocol Operations**
  v2 adds new protocol requests including 'get_bulk' and 'inform' request Pdu's.
- **Security**
- **Manager to Manager Capability**

Traps For UNIX

Trap coexistence with HP OpenView for HPUX. A trap server for stand alone operation.

Asynchronous Mode For UNIX

Async mode is currently only available for MS-Windows.

Demo Engine

SNMP++ demo engine which would allow a local database to be present to simulate an agent. All gets & set would read or write from a local ASCII database. The database could be made up and customized by anyone with knowledge of the agent MIB.

Community Name Database Access

Allow community names to be accessed from a community name database.

SNMP++ Script

Scripting language written using Lex an Yacc for easy SNMP coding.

Oid Database

Allow macro names to be used for referencing Oid's.

Full Win32 Support

Full Win32 support running allowing Win32 network management apps.

Solaris OS Support

Support for Sun Solaris OS.

Apple OS Support

OS/2 Support

NMS Support

Listing and Description of Files

- *oid.h* - Class definition for the Object Identification class.
- *vb.h* - Class definition for the Variable binding class.
- *pdu_cls.h* - Class definition for the PDU Container Class. (**MS-Windows Only**)
- *snmp.h* - Class definition for the SNMP class.
- *snmp_pp.lib* - SNMP++ MS-Windows Win16 Library
- *libsnmp++.a* - SNMP++ HPUX library for HPUX rev 9.X for series 700 & 800 series workstations.

Required Files For MS-Windows Development

oid.h
vb.h
snmp.h
pdu_cls.h
snmp_pp.lib
winsnmp.h, winsnmp.lib, winsnmp.dll.

Required Files For HPUX Development

oid.h
vb.h
snmp.h
libsnmp++.a

References

[Comer]
Comer, Douglas E. , Internetworking with TCP/IP, Principles, Protocols and Architecture, Volume I Prentice Hall, 1991.

[Gama, Helm, Johnson, Vlissides]
Erich Gama, Richard Helm , Ralph Johnson, John Vlissides , Design Patterns, Addison Wesley, 1995.

[Meyers]
Meyers, Steve, Effective C++, Addison Wesley, 1994.

[Petzold]
Petzold Charles, Programming MS-Windows, Microsoft Press

[RFC 1452]
J. Case, K. McCloghrie, M. Rose, S. Waldbusser, Coextistence between version 1 and version 2 of the Internet-standard Network Management Framework, May 03, 1993.

[RFC 1442]
J. Case, K. McCloghrie, M. Rose, S. Waldbusser, Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2), May 03 , 1993.

[Rose]
Rose, Marshall T. , The Simple Book, An Introduction to Internet Management , Second Edition, Prentice Hall Series 1994.

[Rumbaugh]
Rumbaugh, James, Object-Oriented Modeling and Design, Prentice Hall, 1991.

[Saks]
Saks, Dan, C++ Programming Guidelines, Thomas Plum & Dan Sacks, 1992.

[Stallings]
Stallings, William, SNMP, SNMPv2 and CMIP The Practical Guide to Network Management Standards, Addison Wesley, 1993.

[Stroustup]
Stroustrup , Bjarne, The C++ Programming Language, Edition #2 Addison Wesley, 1991.

[WinSNMP]
WinSNMP, Windows SNMP An Open Interface for Programming Network Management Application under Microsoft Windows. Version 1.1.

[WinSockets]
WinSockets, Windows Sockets, An Open Interface for Network Programming under Microsoft Windows.

Appendix A, Public Oid Class Interface:

```
public:

  // constructor using no arguments
  // initialize octet ptr and string
  // ptr to null
  Oid::Oid( void);

  // constructor using a dotted string
 Oid::Oid( const char WINFAR * dotted_oid_string);

  // constructor using another oid object
 Oid::Oid ( const Oid &oid);

  // destructor
 Oid::~Oid();

  // assignment to a string operator overloaded
 Oid::Oid& operator=( const char WINFAR *dotted_oid_string);

  // assignment to another oid object overloaded
 Oid::Oid& operator=( const Oid &oid);

  // equivalence operator overloaded
  friend int operator==( Oid &x,Oid &y);

  // equivalence operator overloaded
  friend int operator==( Oid &x,char WINFAR *dotted_oid_string);

  // append operator, appends a string
 Oid::Oid& operator+=( const char WINFAR *a);

  // appends an int
 Oid::Oid& operator+=( const unsigned long i);

 // return an oid as a dotted string value
  char WINFAR * Oid::strval();

  // return dotted string value from the right
  // where the user specifies how many positions to print
  char WINFAR * Oid::strval( unsigned long n);

  // return a dotted string where the caller specifies
  // where the starting position is and how many to include to the right
  char WINFAR * Oid::strval( unsigned long start, unsigned long n);

// return the WinSnmp oid part
  SmiLPOID Oid::oidval();
```

Appendix A, Public Oid Class Interface Continued:

```
// set the leftmost instance
void Oid::set_instance(unsigned long i);

// modify position n of an oid to value i
// indexes are 1 to n
void Oid::set_instance( unsigned long n,    // instance # to change
                         unsigned long i);   // new value

// returns the rightmost value of the oid
unsigned long Oid::get_instance();

// returns the value of an oid
// at position n
unsigned long Oid::get_instance( unsigned long n);

// return the len of the oid
unsigned long Oid::len();

// trim off the n leftmost values of an oid
// Note!, does not adjust actual space for
// speed
void Oid::trim( unsigned long n);

// compare the n leftmost bytes
// returns TRUE or FALSE
int Oid::nCompare( unsigned long n, const Oid &o);
```

Appendix B, Public Vb Class Interface:

```
public:
  //-----[ constructors / destructors ]-------------------------------

  // constructor with no arguments
  // makes an vb, unitialized
  Vb::Vb( void);

  // constructor to initialize the oid
  // makes a vb with oid portion initialized
 Vb::Vb( const Oid oid);

  // destructor
  // if the vb has a oid or an octect string then
 Vb::~Vb();

  //-----[ set oid / get oid ]------------------------------------------

   // set value oid only with another oid
   void Vb::set_oid( const Oid &oid);

   // set oid value with a const string
   void Vb::set_oid( const char WINFAR * dotted_oid_string);

   // get oid portion
   void Vb::get_oid( Oid &oid);

//-----[ set value ]--------------------------------------------------

   // set the value with an int
   void Vb::set_value( int i);

   // set the value with a long signed int
   void Vb::set_value( long int i);

   // set the value with an unsigned long int
   void Vb::set_value( unsigned long int i);

   // set value for building an 64 bit counter
   void Vb::set_value( unsigned long int hi,unsigned long int lo);

   // set value for setting an oid
   // creates own space for an oid which
   void Vb::set_value( Oid &varoid);

   // set value for setting an octet string
   // creates own space for a octet
   void Vb::set_value( unsigned char WINFAR * ptr, unsigned long len);
```

Appendix B, Public Vb Class Interface Continued:

```
  // set value on a string
  // makes the string an octet
  // this must be a null terminates string
  void Vb::set_value( char WINFAR * ptr);

  // set an ip address as a value
  void Vb::set_value ( Iptype ipaddr);

  // set value for timeticks
  void Vb::set_value( TimeTicks timeticks);

  // set value for a counter
  void Vb::set_value( Counter counter);

  // set value for a gauge
  void Vb::set_value( Gauge gauge);

//----[ get value ]---------------------------------------------

  // get value int
  // returns 0 on success and value
  int Vb::get_value( int &i);

  // get the signed long int
  // returns 0 on success and a value
  int Vb::get_value( long int &i);

  // get the unsigned long int
  // returns 0 on success and a value
  int Vb::get_value( unsigned long int &i);

  // get value for building an 64 bit counter
  // returns 0 on success and a value
  int Vb::get_value( unsigned long int &hi,unsigned long int &lo);

  // get the oid value
  // free the existing oid value
  // copy in the new oid value
  int Vb::get_value( Oid &varoid);

  // get a unsigned char string value
  // destructive, copies into given ptr
  // also returned is the len length
  // Note! the caller must provide a target string big
  // enough to handle the vb string
  int Vb::get_value( unsigned char WINFAR * ptr, unsigned long &len);

  // get an unsigned char array
  // caller specifies max len of target space
  int Vb::get_value( unsigned char WINFAR * ptr,  // pointer to target space
                     unsigned long &len,           // returned len
                     unsigned long maxlen);        // max len of target space
```

Appendix B, Public Vb Class Interface Continued:

```
// get a char * from an octet string
// the user must provide space or
// memory will be stepped on
int Vb::get_value( char WINFAR *ptr);

// get an ip address
int Vb::get_value( Iptype ipaddr);

// get value for timeticks
int Vb::get_value( TimeTicks &timeticks);

// get value for a counter
int Vb::get_value( Counter &counter);

// get value for a gauge
int Vb::get_value( Gauge &gauge);


//-----[ misc]-------------------------------------------------------

// return the current syntax
// This method violates Object Orientation but may be useful if
// the caller has a vb object and does not know what it is.
// This would be useful in the implementation of a browser.
SmiUINT32 Vb::get_syntax();


// return smivalue portion
// this should really be a friend function
// since it violates data hiding
SmiLPVALUE Vb::get_smival();
```

Appendix C, Public SNMP Class Interface:

```
public:

  //------------------[ constructor,blocked usage ]---------------------
  Snmp::Snmp( WORD hwnd,                    // parent handle
              Protocol protocol             // type of protocol to use
              const char *community_name,  // community name
              int *status);                 // construction status

  //------------------[ constructor, async usage]-----------------------
 Snmp:: Snmp( WORD hwnd,                    // parent handle
              Protocol protocol,            // type of protocol to use
              const char *community_name, // community name
              snmp_callback lpcallback,     // callback function address
              int *status);                 // construction status

  //-------------------[ destructor ]-----------------------------------
 Snmp::~Snmp();

  //--------------------[ set timeout ]---------------------------------
  void Snmp::set_timeout( DWORD t);

  //---------------------[ get timeout ]--------------------------------
  DWORD Snmp::get_timeout();

  //----------------------[ set retry ]---------------------------------
  void Snmp::set_retry( int r );

  //----------------------[ get retry ]---------------------------------
  int Snmp::get_retry( );

  //----------------------[ set the community name ]--------------------
  void Snmp::set_community( const char * new_name);

  //----------------------[ set community name ptr and len ]
  void Snmp::set_community( const char * new_name, int len);

  //----------------------[ set partial shut down value ]----------------
  void Snmp::set_shutdown_val( WORD val) { partial_shutdown_val = val;};

//------------------------[ get ]--------------------------------------
  int Snmp::get( Vb *vb,                  // pointer to array of vb objects
                 int vb_count,            // count of vb objects
                 long int &err_status,    // returned error status
                 long int &err_index,     // returned error index
                 const char *dest_addr);  // get address
```

Appendix C, Public SNMP Class Interface, Continued:

```
//------------------------[ get async ]----------------------------------
  int Snmp::get_async( Vb *vb,                     // pointer to array of vb objects
                       int vb_count,               // count of vb objects
                       long int req_id,            // request id to use, returned
                       const char *dest_addr);     // address to send to

  //------------------------[ get next ]-----------------------------------
  int Snmp::get_next(Vb *vb,                       // pointer to array of vb objects
                     int vb_count,                 // count of vb objects
                     long int &err_status,         // returned error status
                     long int &err_index,          // returned error index
                     const char *dest_addr) ; // get address

  //------------------------[ get next async ]-----------------------------
  int Snmp::get_next_async( Vb *vb,                // pointer to array of vb objects
                            int vb_count,          // count of vb objects
                            long int req_id,       // request id to use, returned
                            const char *dest_addr);  // address to send to

  //-------------------------[ set ]---------------------------------------
  int Snmp::set( Vb *vb,                           // pointer to array of objects
                 int vb_count,                     // count of objects
                 long int &err_status,             // returned error status
                 long int &err_index,              // returned error index
                 const char *dest_addr) ; // target address

  //------------------------[ set async ]----------------------------------
  int Snmp::set_async( Vb *vb,                     // pointer to array of vb objects
                       int vb_count,               // count of vb objects
                       long int req_id,            // request id to use, returned
                       const char *dest_addr);     // address to send to

//---------------------------[ get many ,ints ]--------------------------
  int Snmp::get_many( Oid base_oid,                // base oid to use
                      int *ret_values,             // returned int values
                      int num_values,              // number of values to get
                      long int &err_status,        // returned error status
                      long int &err_index,         // returned error index
                      const char *dest_addr);      // target address

  //---------------------------[ get many ,long ints ]---------------------
  int Snmp::get_many( Oid base_oid,                // base oid to use
                      long int *ret_values,        // returned int values
                      int num_values,              // number of values to get
                      long int &err_status,        // returned error status
                      long int &err_index,         // returned error index
                      const char *dest_addr);      // target address
```

Appendix C, Public SNMP Class Interface, Continued:

```
//---------------------------[ get many ,ip addresses ]------------------
  int Snmp::get_many( Oid base_oid,              // base oid to use
                      Iptype ret_values[],       // returned int values
                      int num_values,            // number of values to get
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr); // target address

  //---------------------------[ get many array of chars ]-----------------
  int Snmp::get_many( Oid base_oid,              // base oid to use
                      unsigned char ret_values[][MAX_ADDR_LEN],
                      int num_values,            // number of values to get
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr);  // target address

  //---------------------------[ set many ,ints ]--------------------------
  int Snmp::set_many( Oid base_oid,              // base oid to use
                      int *in_values,            // values to set
                      int num_values,            // number of values to set
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr);  // target address

  //---------------------------[ set many ,long ints ]---------------------
  int Snmp::set_many( Oid base_oid,              // base oid to use
                      long int *in_values,       // values to set
                      int num_values,            // number of values to set
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr);  // target address

  //---------------------------[ set many ,iptypes  ]---------------------
  int Snmp::set_many( Oid base_oid,              // base oid to use
                      Iptype in_values[],        // values to set
                      int num_values,            // number of values to set
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr);  // target address

  //---------------------------[ set many ,uinsigned chars  ]-------------
  int Snmp::set_many( Oid base_oid,              // base oid to use
                      unsigned char in_values[][MAX_ADDR_LEN], // values to set
                      int num_values,            // number of values to set
                      long int &err_status,      // returned error status
                      long int &err_index,       // returned error index
                      const char *dest_addr);  // target address
```

Appendix C, Public SNMP Class Interface, Continued:

```
//----------------------------[ trap register ]--------------------------
  // To be used only for async Snmp objects
  // Note, WinSnmp v1 informs the trap receiver with SnmpV2 type
  // trap messages. In SnmpV2, the trap format consists of N variable
  // bindings where...
  // - the first vb is the time stamp
  // - the second vb identifies the trap
  // - the third through N vb's are the payload
  //
  //
  int Snmp::trap_register( const char * mgr_addr,    // managers source address
                           const char * agent_addr,  // agents address
                           Oid oidmask,              // oid mask or empty for all
                           int on);                  // boolean for on or off
```

Appendix D, Public Timeticks, Counter and Gauge Class Interface:

```
public:
    // constructor with no args
   TimeTicks::TimeTicks( void)

    // constructor with an unsigned long
    TimeTicks::TimeTicks( unsigned long i)

    // overloaded equivalence operator to an unsigned long
   TimeTicks::TimeTicks& operator=(unsigned long int i)

    // overloaded equivalence operator to another Timeticks object
    TimeTicks::TimeTicks& operator=(const TimeTicks &uli)

    // behavior like an unsigned long int
    operator Counter::unsigned long()

  public:
    // construct a counter object
    Counter::Counter( void)

    // construct a counter with an unsigned long
    Counter::Counter( unsigned long i)

    // overloaded assignment to an unsigned long int
    Counter::Counter& operator=(unsigned long int i)

    // overloaded assignment to another counter object
    Counter& operator=(const Counter &uli)

    // give it the full functionality of an unsigned int
    operator Counter::unsigned long()

 public:
    // constructor
    Gauge::Gauge( void)

    // constructor with an unsigned long
    Gauge::Gauge( unsigned long i)

    // overloaded assignment with an unsigned long
    Gauge::Gauge& operator=(unsigned long int i)

    // overloaded assignment with another gauge
    Gauge::Gauge& operator=(const Gauge &uli)

    // give it full functionality of an unsigned long
    operator Gauge::unsigned long()
```