The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS℠
VOLUME 3, NUMBER 2 AUGUST, 1994

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol (SNMP). In each issue, ***The Simple Times*** presents: a refereed technical article, an industry comment, and several featured columns. In addition, some issues include brief announcements, summaries of recent publications, and an activities calendar. For information on submissions, see page 12.

In this Issue:

Technical Article

Kaj Tesink, Bellcore and Ted Brunner, Tektronix

In this issue: *(Re)Configuration of ATM Virtual Connections with SNMP*

The IESG has recently approved a MIB module for the management of ATM interfaces, RFC 1695. This MIB module allows for the configuration of *virtual connections* (VCs). This article explains by example how this MIB module must be used in order to configure, reconfigure, or release VCs. While this procedure seems elaborate, it actually allows for detailed, step-by-step error checking for a VC (re)configuration in a simply scripted management application.

Introduction

For this explanation, the following ATM concepts are relevant:

- an ATM interface may support multiple VCs simultaneously;
- the portion of a VC between two adjacent ATM interfaces is called a *virtual link* (VL);
- ATM *intermediate systems* (ISs, e.g., switches, networks) "cross-connect" VLs to form segments of VCs;
- a VC may traverse multiple segments;
- ATM hosts complete VCs by linking them to the appropriate user application or higher layer protocol;
- ATM interfaces may distinguish a two level multiplexing hierarchy;
- the lowest level interface distinguishes *virtual path links* (VPLs);
- a VPL may support multiple *virtual channel links* (VCLs);
- two different types of VCs may be formed: *virtual path connections* (VPCs) and *virtual channel connections* (VCCs);

- VPLs are identified by *virtual path identifiers* (VPIs), and VCLs are identified by *virtual channel identifiers* (both VPIs and VCIs); and,
- A VC is characterized by: a traffic pattern and *quality of service* (QoS), and, a topology; this may be a point-to-point, a point-to-multipoint or a multipoint-to-multipoint topology.

The ATM MIB treats VCCs and VPCs in much the same way. Thus, in the interest of simplicity this article describes procedures in the general terms of VCs and VLs.

The ATM MIB applies to ATM hosts, to ATM ISs and to ATM service providers who present an "ATM cloud". Because of their different roles, some portions of the MIB do not apply to all systems, e.g., the tables describing VL cross-connects only apply to intermediate systems and to ATM service.

Procedure summary

The manipulation of VCs can be broken down into several phases, each of which affects different portions of the end-to-end VC:

VC establishment consists of the following phases:

- reserve appropriate VLs;
- characterize traffic on the VLs; and,
- cross-connect the VLs in ISs, and associate the VLs with a user application in the hosts.

whilst VC release consists of the following phases:

- release cross-connect in ISs; and,
- release all VLs.

The order of the phases is important to the procedure. Although the whole procedure may seem complex, each step serves an important function in detecting errors that my occur during VC setup and takedown. Experience with the SNMP `set` operation has shown that they are, in general, more difficult than the `get` operation, and that detailing error conditions facilitates debugging.

This article illustrates the above procedure through an example of the configuration of a VCC between "HostA" and "HostB" through one IS. It assumes that the underlying VPLs are already established:

```
Device       Interface       ATM VC id
=========    ==========      =============
HostAToIS    ifIndex=11      VPI/VCI=17,19
ISToHostA    ifIndex=13      VPI/VCI=17,19
ISToHostB    ifIndex=3       VPI/VCI=5,7
HostBToIS    ifIndex=1       VPI/VCI=5,7
```

VC establishment

The first phase is to reserve the appropriate VLs.

Basic interface configuration information is provided through the

```
atmInterfaceConfTable
```

This table provides some VC constraints such as the maximum number of VCs that can be supported (`atmInterfaceMaxVpcs/Vccs`), the number of active VCs (`atmInterfaceVpcs/Vccs`), and the range of appropriate VC identifiers (`atmInterfaceMaxActiveVpi/VciBits`).

The management application creates a VL entry in the VL table (`atmVclTable`) by setting the row status to `createAndWait`. Note that the VC indices are chosen by the manager, and not by the agent. The index clause of the `atmVclTable` is

```
{ ifIndex, atmVclVpi, atmVclVci }
```

Thus:

```
HostA  set atmVclRowStatus.11.17.19=5

IS     set atmVclRowStatus.13.17.19=5
       set atmVclRowStatus.3.5.7=5

HostB  set atmVclRowStatus.1.5.7=5
```

This fails (on a particular ATM interface) if:

- the agent supports read-only operations on this table only (i.e., VC (re)configuration is not allowed);
- the selected `ifIndex` value does not exist, or is not an ATM interface (see `ifTable`);
- the maximum number of VCs supported for this interface has been reached (`atmInterfaceMaxVpcs/Vccs`);
- the selected VPI/VCI values are unavailable for use (see `atmInterfaceMaxActiveVpi/VciBits`); or,
- The selected VPI/VCI values are in use or reserved (see existing rows in the `atmVclTable`).

Otherwise, the agent creates a row and reserves the VPI/VCI values on that port. It also increments `atmInterfaceVpcs/Vccs`.

The second phase is to characterize traffic on the VLs.

The ATM MIB contains a table

```
atmTrafficDescrParamTable
```

that contains vectors with traffic parameter values. The ATM VL tables characterize the traffic for the transmit and receive direction by pointing to the appropriate entries in the `atmTrafficDescrParamTable`. Multiple

VLs in the atmVclTable/atmVplTable can point to the same vector in the atmTrafficDescrParamTable. This technique allows the agent to predefine self-consistent traffic vectors in the atmTrafficDescrParamTable. In addition, the agent may support read-create access, allowing the manager to specify additional vectors.

The traffic vector consists of an atmTrafficDescrType column, describing the type of traffic, five columns that parameterize the traffic (atmTrafficDescrParam1 through atmTrafficDescrParam5 — the number of columns actually used depends on the traffic type), and a column indicating the quality of service (atmTrafficQoSClass). The table is completed by a row status column (atmTrafficDescrRowStatus) and the atmTrafficDescrParamIndex which indexes the table.

Thus, the manager selects an existing row(s) in the atmTrafficDescrParamTable, or, if no suitable row(s) exists, the manager must create a new row(s) in that table. For example, when the next free row is 100:

```
set atmTrafficDescrRowStatus.100
        =creatAndWait
```

This action fails if:

- the agent does not support read-create on this table, for example, because only a (pre)fixed set of traffic characterizations are supported; or,
- the specified row is already active.

As an example, a "best effort" VL where none of the traffic parameters is used is characterized by:

```
set atmTrafficDescrType.100
        =atmNoTrafficDescriptor
```

As another example, an ATM VC with a 64kbps peak rate is characterized by:

```
set atmTrafficDescrType.100=atmNoClpNoScr
    atmTrafficDescrParam1.100=167
```

(peak cell rate = 64000bps/(48bytes x 8bits), or 167cells/sec.)

The quality of service class is specified by, e.g,

```
set atmTrafficQoSClass.100=bestEffort
```

These actions fail if:

- the parameters are mutually inconsistent; or,
- the agent does not support the requested values.

The manager activates the traffic descriptor parameter row by:

```
set atmTrafficDescrRowStatus.100=active
```

The management application now characterizes the traffic parameters of all the VLs associated with the VC by pointing the receive and transmit traffic index (atmVpl/VclReceiveTrafficDescrIndex and atmVpl/VclTransmitTrafficDescrIndex) in the VL table

```
atmVpl/VclTable
```

to the

```
atmTrafficDescrParamTable
```

rows containing desired ATM traffic parameter values. Thus:

```
HostA set
  atmVclReceiveTrafficDescrIndex.11.17.19
      =100
  atmVclTransmitTrafficDescrIndex.11.17.19
      =100

IS set
  atmVclReceiveTrafficDescrIndex.13.17.19
      =100
  atmVclTransmitTrafficDescrIndex.13.17.19
      =100
  atmVclReceiveTrafficDescrIndex.3.5.7
      =100
  atmVclTransmitTrafficDescrIndex.3.5.7
      =100

HostB set
  atmVclReceiveTrafficDescrIndex.1.5.7
      =100
  atmVclTransmitTrafficDescrIndex.1.5.7
      =100
```

This action fails (on a particular VL) if:

- insufficient resources are available.

The manager now activates the VLs by setting the row status (atmVpl/VclRowStatus) to active, thus:

```
HostA  set atmVclRowStatus.11.17.19=active

IS     set atmVclRowStatus.13.17.19=active
           atmVclRowStatus.3.5.7=active

HostB  set atmVclRowStatus.1.5.7=active
```

If this set is successful, the agent has reserved the resources to satisfy the requested traffic parameter values and the QoS Class for that VL.

The third and final phase is to cross-connect the VLs in the intermediate systems.

On the IS, the atmVcCrossConnectTable must be used to cross-connect the VLs. The atmVpl/VclTables

have a cross-connect identifier column for this purpose (atmVpl/VclCrossConnectIdentifier). Different rows in the atmVpl/VclTable that have the same cross-connect identifier value are cross-connected. This is achieved through cross-connect tables (atmVp/VcCrossConnectTable).

The first step to cross-connecting VLs is to obtain a unique cross-connect index. An object,

```
atmVp/VcCrossConnectIndexNext
```

is defined for this purpose. A get-next will obtain a value (e.g., 3333). The object is defined such that after a retrieval operation the agent will increment the value to the next unassigned one.

This operation fails if:

- the value 0 is returned, which means that all available values are in use (e.g., the switch can not support more VCs).

The second step to cross-connecting VLs has the manager connect VCLs through creation of a row in the atmVcCrossConnectTable, which is indexed by:

```
{ atmVcCrossConnectIndex,
  atmVcCrossConnectLowIfIndex,
  atmVcCrossConnectLowVpi,
  atmVcCrossConnectLowVci,
  atmVcCrossConnectHighIfIndex,
  atmVcCrossConnectHighVpi,
  atmVcCrossConnectHighVci }
```

(Cross-connecting VPLs works in the same way — only the INDEX clause differs.) The VL indices for the interface with the lowest index value (i.e., ifIndex=3) must be specified first in the table's index. Thus:

```
set atmVcCrossRowStatus.3333.3.5.7.13.17.19
        =createAndWait
```

This request fails (on a particular cross-connect) if:

- the specified Low-index value is higher than that of the High-index;
- the requested topology is not supported by the agent (e.g., a multipoint VC may be requested that is not supported by this IS);
- the traffic and QoS parameter values of the specified VCLs are mutually incompatible; or,
- The agent may be unable to connect the two VCLs (e.g., no resources are available — note that the agent may represent a network).

If the request is successful the agent fills in the atmVcCrossConnectIndex values in the corresponding atmVclTable rows, i.e.,

```
atmVclCrossConnectIdentifier.13.17.19
    = 3333

atmVclCrossConnectIdentifier.3.5.7
    = 3333
```

The manager can now activate the cross-connect row by:

```
set
  atmVcCrossConnectRowStatus.3333.3.5.7.13.17.19
      =active
```

This request fails (on a particular cross-connect) if:

- the agent cannot reserve appropriate resources for this cross-connect.

Finally, the manager turns on the traffic through the cross-connected VLs by:

```
set
atmVcCrossConnectAdminStatus.3333.3.5.7.13.17.19
        =up
```

At this point the traffic flow must be actually turned on. In the hosts the link to the application must be made.

```
HostA  set atmVclAalType.11.17.19=aal1

HostB  set atmVclAalType.1.5.7=aal1
```

This action fails if:

- the requested application and VL traffic pattern do not match.

Finally, the manager turns on the traffic at the host by:

```
HostA  set atmVclAdminStatus.11.17.19=up

HostB  set atmVclAdminStatus.1.5.7=up
```

This step-by-step process above can be shortened by using the createAndGo value for the row-status objects. However, the advantage of detailed step-wise error checking would be lost. Therefore, the step-wise process is recommended.

Graceful VC Release

The first phase is to release the cross-connects in the IS.

To release a VC, all cross-connects and associated VLs must be released by setting the row status of the associated table entries to destroy. Thus, for IS:

```
set
  atmVcCrossConnectRowStatus.3333.3.5.7.13.17.19
      =destroy
```

Removal of all rows with

```
atmVcCrossConnectRowStatus.3333.*
```

will free the value 3333 for future use by the

`atmVcCrossConnectIndexNext`

and the

`atmVclCrossConnectIdentifier`

values will be removed from the associated VLs, to signify that they are no longer cross-connected. Cross-connect resources are released.

The second phase is to release the VLs.

To reclaim the VLs associated with the VC, each associated table entry must be destroyed. Thus:

```
HostA  set atmVclRowStatus.11.17.19=destroy

IS     set atmVclRowStatus.13.17.19=destroy
           atmVclRowStatus.3.5.7=destroy

HostB  set atmVclRowStatus.1.5.7=destroy
```

Upon these actions the agents will release the associated VL resources, and decrement `atmInterfaceVpcs/Vccs`.

It is recommended to release a cross-connect before releasing the individual VLs. The reason is that releasing a VL first may, in some implementations, be interpreted as a request for a configuration change (e.g., a multipoint topology where one leaf is being deleted; see below). Proper agent implementation should release cross-connects automatically if:

- a VL is released and cross-connect reconfiguration is not supported by the agent; or,
- a VL is released and the remaining topology is meaningless to the agent (e.g., one of two cross-connected VLs is released).

The third phase is to release the traffic descriptors.

To release the traffic parameter values associated with the transmit and receive directions of the VLs, the rows of the traffic descriptor table (`atmTrafficDescrParamTable`) pointed to by the VLs, must be deleted. Deletion proceeds in the normal way with `atmTrafficDescrRowStatus`. Such a deletion fails if:

- the agent does not support read-write access to this table; or,
- the traffic parameters of this row are still used by another VC.

VC Reconfiguration

Several VC reconfiguration applications are detailed below. Several require additional capabilities which an agent may support.

Traffic and/or QoS parameter value changes. These do not require additional agent capabilities. The manager takes down the current VC, defines new VLs with the desired parameters, and brings up the new VC following the rules outlined above. This is most simply done as an entirely new set of VLs. If there is a desire to retain the VPI/VCI values, the manager may follow these steps:

- turn VL traffic off at hosts
  (set the `atmVclAdminStatus` to `down`);
- release the cross-connect at ISs
  (set the `atmVcCrossConnectRowStatus` to `destroy`);
- turn VL traffic off at ISs
  (set the `atmVclAdminStatus` to `down`);
- take the VLs out of service at the hosts and ISs
  (set the `atmVclRowStatus` to `notInService`), then, as before, configure the VLs, cross-connect, and turn traffic on;
- find or create the new traffic parameter row(s);
- associate the VLs with the new traffic parameter row(s);
- activate the VLs at the hosts and ISs
  (set the `atmVcpRowStatus` to `active`);
- turn VL traffic on at ISs
  (set the `atmVcVrossConnectAdminStatus` to `up`); and,
- turn VL traffic on at hosts
  (set the `atmVclAdminStatus` to `up`).

Changing the VC application at the hosts (for example, changing the AAL), requires additional agent capabilities.

A VC topology change requires additional agent capabilities. To accomplish a point to multipoint leaf addition, follow these steps:

- define the VL to be added; and,
- define an additional row in the cross-connect table

To accomplish a destination change, follow these steps:

- delete the appropriate row in the cross-connect table;
- delete the appropriate VL (host and switch);
- define the VL to be added; and,
- define an additional row in the cross-connect table.

Tracing of VCs

In order to trace a VC through multiple switches and hosts, a manager needs to refer to information about how switches and hosts are interconnected. This includes both the location of neighboring switches and hosts, and the topology of the links to neighboring switches and hosts. We call this the neighbor information. The "location" of a switch or host, for our purposes, is the address of its SNMP agent. The topology of links between switches is captured in a mapping of each local interface to a neighbor switch or host, and to an interface on that neighbor. This latter need results from a topology ambiguity when switches have several parallel interconnecting links: which interface connects to which? Thus, interface values are needed on both sides of the link.

The neighbor information is expressed, for each physical interface on a switch, as the address of the agent on the neighbor switch to which this interface is connected, and the interface name on the neighbor switch to which this interface is connected.

```
ifIndex -> neighborAddress, neighborIfName
```

Populating the neighbor information objects requires quite specific topological information. It can be manually configured, or auto-discovered. In the latter case, an agent acquires the information by interchanging with other agents its address and its interfaces. The ATM Forum has defined an ILMI MIB and access method which can be used for such a purpose. The ILMI messages are interchanged on a single ATM link, and can access the ILMI MIB objects. Using this, each agent sends an ILMI query out each of its physical interfaces, requesting its neighbor's agent address and the incoming interface name. Upon receiving such a query, a switch must answer with its address and the name of the interface on which the query came in. Full network auto-discovery requires all switches to participate.

The ATM Forum's Network Management Subcommittee has recently approved two new ILMI objects which will support this auto-discovery. They are: `atmfMyIpNmAddress` (non-tabular) and `atmfPortMyIfName` in the `atmfPortTable`, which is indexed by physical interface.

In order to represent this neighbor information at the SNMP level, the ATM MIB contains two objects. They are named: `atmInterfaceMyNeighborIpAddress` and `atmInterfaceMyNeighborIfName`, both in the `atmInterfaceConfTable`, which is indexed by `ifIndex`. Thus the neighbor information is:

```
ifIndex
    -> atmInterfaceMyNeighborIpAddress,
       atmInterfaceMyNeighborIfName
```

To illustrate this, consider the neighbor information for our example, which is maintained by the agents residing on HostA, on the IS, and on HostB:

```
HostA  index11 -> ipAddrIS, ifName13

IS     index13 -> ipAddrA,  ifName11
       index3  -> ipAddrB,  ifName1

HostB  index1  -> ipAddrIS, ifName3
```

To trace a VC, the manager follows a simple algorithm involving the `atmVc/VpCrossConnectTable` and the neighbor information in the `atmInterfaceConfTable`. To start, it identifies a starting point switch or host, a VC and a direction, probably by naming an outgoing interface and VPI/VCI values. The outgoing VPI/VCI values equal the incoming VPI/VCI values on the next hop interface. From the `atmInterfaceConfTable` it determines the next hop switch, and the incoming interface on that switch. From the `atmVc/VpCrossConnectTable` on the next hop switch it determines the outgoing interface, and the outgoing VPI/VCI values.

The algorithm is repeated until the VC terminates at a host. Since it has no cross-connections, a host does not support the `atmVc/VpCrossConnectTable`.

Industry Comment

Marshall T. Rose

It's official: this is the last issue of ***The Simple Times*** for 1994. The reason is simple: the editorial staff (me) just doesn't have the time to put in on each issue. Look for the third volume at the beginning of 1995.

Security and Protocols

Keith McCloghrie

The ATM Forum is a consortium of member companies and organizations. Its initial charter was to produce "interoperability agreements". These agreements are designed to fill in the gaps where no standards are defined. As it has turned out, several of the Forum's specifications have in fact defined new standards. One such new standard is the Interim Local Management Interface (ILMI).

The purpose of the ILMI is to allow the user-side and network-side of a User Network Interface (UNI) to communicate with each other in order to exchange management information concerning the local ATM interface. Rather than defining a new protocol, the ILMI is defined to use the same message formats and semantics as are used by SNMPv1. Thus, it has the

same five PDU-types as SNMPv1 (`get`, `get-next`, `set`, `get-response`, and, `trap`), and the variables included in these PDUs are defined in MIBs using SNMPv1's SMI.

However, it is most important to realize that the ILMI has a different paradigm from the use of SNMP for network management. In network management, a manager issues requests and an agent responds with responses and generates traps. In contrast, the ILMI is used for interface management between the two UNI management entities (UMEs), one on either side of the UNI; **both** UMEs can and do send requests as well as responses and traps; the two UMEs have their own (potentially different) values for the same MIB objects.

The term "Interim" was chosen in the early days of the ATM Forum, when there was great reluctance to deviate from ITU-T (formerly the CCITT) standards. However, the ITU-T had no suitable protocol defined for interface management, and thus the ILMI was adopted in the "interim". With the passage of time and the expanding use of the ILMI, it is now joked that it would make more sense if the "I" stood for "indefinite".

The ILMI in UNI 2.0

The initial MIB for use with the ILMI was defined in the version 2.0 specification of the UNI. This MIB contained:

- configuration parameters, such as the maximum number of connections supported by the UME, and whether the interface is a Public or Private UNI;
- tables listing the Permanent Virtual Connections (one for VPCs, and another for VCCs), giving operational status, traffic parameters, and, QoS class for each connection;
- a few optional statistics values for cells received and transmitted; and,
- values for the transmission and media types of the interface.

All of the above were defined to be read-only.

In addition to the ILMI MIB, a UME was expected to support the `system` group from MIB-II, `coldStart`, `linkUp` and `linkDown` traps, and allowed to support enterprise-specific MIB extensions.

Address Registration

As well as some clean-up of the version 2.0 definitions, the major addition in the version 3.0 specification was support for the registration of ATM addresses. The UNI version 3.0 specification defined the format of ATM addresses (for use with the UNI signaling protocol) as having the syntax and structure of OSI NSAP addresses, i.e., being the concatenation of a network prefix with an *End System Identifier* (ESI). The ESI is normally an IEEE MAC address, and the network prefix is the same on all ports of an ATM switch.

The use of the ILMI to perform address registration specifies that the network-side UME issues a SetRequest to provide the host with the network prefix, so that the user-side UME can combine the prefix and its own MAC address, and then issue its own SetRequest for the combined value. Thus, these address registration procedures allow both the host and the switch to obtain the host's ATM address(es) merely from having the switch know the network prefix and the host know its MAC address(es).

Support for Topology Discovery

The recently completed UNI 3.1 specification has two small but significant additions to the ILMI. One is support for ATM network topology discovery. This support is in the form of MIB objects giving the network-layer address at which the local system (host or switch) receives network management requests. So, for example, a switch which is managed by SNMP over UDP would contain its IP address in a ILMI MIB object. The inter-connected UME would read that ILMI MIB object and make the value available to its NMS through its network management MIB. By this means, an NMS can query one device to ascertain the address of its neighbor. Another ILMI MIB object provides the means to differentiate between multiple links between the same two devices, by containing the textual name (e.g., the value of `ifName`) of an interface.

Auto-Configuration of UNI Version

The other ILMI addition in the UNI 3.1 specification is an indication of the highest version of the UNI supported. Unfortunately, there's an incompatibility in the link-layer of the signaling protocol between versions 3.0 and 3.1 which prevents interoperability. This addition to the ILMI MIB allows each UME to inspect the highest version of the UNI specification supported by the other. Thus, a system which supports both 3.0 and 3.1 can automatically configure itself to interoperate its neighbor.

Future Uses

Other working groups of the ATM Forum's Technical Committee are also looking to use the ILMI for initialization purposes. Specifically, a contribution has been accepted for ILMI additions allowing a host to obtain

the ATM address of a Service. The LAN-Emulation specification will use this as one means by which a LAN-Emulation Client can reach the LAN-Emulation Configuration Service.

It is not yet determined whether the ILMI will be used at the NNI (the interface between two ATM switches), although it has been suggested as a means of extending the Topology Discovery to cover the whole network. Another possibility is for a switch to be able to automatically distinguish a UNI from an NNI, and perform other auto-configuration.

Summary

Multiple benefits ensue from the choice of using SNMP message formats and semantics for the ILMI. One is that the commands are simple, well-understood by implementors, and code is already available to parse the messages. Another is illustrated in the above descriptions of how incremental additions of a few MIB objects can provide significant increases in functionality, without requiring new messages to be added to the protocol.

Standards

David T. Perkins

In the extended time between the last and current issues there have been eleven new RFCs published containing MIB modules. Four of these were updates to existing MIB moduless. These were RFC 1643, the Ether-Like MIB, which is now a full standard; RFC 1658, the character device MIB; RFC 1659, the RS-232 interface type MIB; and, RFC 1660, the Parallel Printer interface type MIB; which are all now Draft standards.

The seven other MIB moduless reflect work that crosses the spectrum of areas where management is now possible. These include RFC 1595, the SONET/SDH interface type MIB; RFC 1604, the Frame Relay Service MIB; RFCs 1611 and 1612, the DNS Server and Resolver MIBs; RFC 1628, the UPS MIB; RFC 1650, the SNMPv2 version of the Ether-Lime MIB; RFC 1657, the MIB for BGP version 4; RFC 1666, the SNA NAU MIB. RFC 1694, the SMDS Interface Protocol (SIP) Interface Type MIB; RFC 1695, the ATM MIB; RFC 1696, the Modem MIB; and, and, RFC 1697, the Relational Database Management System MIB.

The SNMPv2 SMI

In July of last year, the Network Management Area Director established a transition policy from using the SNMPv1 SMI to using the SNMPv2 SMI. It requires that all new MIB modules be written using the SNMPv2 SMI, except that the new syntax types of `BIT STRING`, `NsapAddress`, `Counter64`, and, `UInteger32` are not allowed except for special cases approved by the Area Directory. After the SNMPv2 SMI is elevated to a Draft standard, all syntax types may be used. However, existing MIB moduless moving to the full standard level, must be written in the SNMPv1 SMI format until the SNMPv2 SMI is elevated to Full standard status.

Use of the SNMPv2 SMI has been much appreciated since it allows then to be much more precise in specifying in parsable form the grouping and rules for compliance to the MIB.

Two MIB module checkers are currently available via e-mail:

```
mib-checker@epilogue.com
mosy@dbc.mtview.ca.us
```

For both, the body of the email message should contain the MIB module to be checked (the “Subject:” lines are not evaluated). Hopefully other checkers will soon be available.

Summary of Standards

SNMPv1 Framework (Full Standards):

- 1155 - Structure of Management Information (SMI);
- 1157 - Simple Network Management Protocol (SNMP);
- 1212 - Concise MIB definitions; and,
- 1213 - Management Information Base (MIB-II).

SNMPv2 Framework (Proposed Standards):

- 1441 - Introduction to SNMPv2;
- 1442 - SMI for SNMPv2;
- 1443 - Textual Conventions for SNMPv2;
- 1444 - Conformance Statements for SNMPv2;
- 1445 - Administrative Model for SNMPv2;
- 1446 - Security Protocols for SNMPv2;
- 1447 - Party MIB for SNMPv2;
- 1448 - Protocol Operations for SNMPv2;
- 1449 - Transport Mappings for SNMPv2;
- 1450 - MIB for SNMPv2;
- 1451 - Manager-to-Manager MIB; and,

- 1452 - Coexistence between SNMPv1 and SNMPv2.

Full Standards:

- 1213 - Management Information Base (MIB-II); and,
- 1643 - Ether-Like Interface Type (SNMPv1).

Draft Standards:

- 1493 - Bridge MIB; and,
- 1516 - IEEE 802.3 Repeater MIB;
- 1559 - DECnet phase IV MIB;
- 1658 - Character Device MIB;
- 1659 - RS-232 Interface Type MIB; and,
- 1660 - Parallel Printer Interface Type MIB.

Proposed Standards:

- 1231 - IEEE 802.5 Token Ring Interface Type MIB;
- 1239 - Reassignment of experimental MIBs to standard MIBs;
- 1243 - AppleTalk MIB;
- 1253 - OSPF version 2 MIB;
- 1269 - BGP version 3 MIB;
- 1271 - Remote LAN Monitoring MIB;
- 1285 - FDDI Interface Type (SMT 6.2) MIB;
- 1315 - Frame Relay DTE Interface Type MIB;
- 1354 - IP Forwarding Table MIB;
- 1381 - X.25 LAPB MIB;
- 1382 - X.25 PLP MIB;
- 1389 - RIPv2 MIB;
- 1406 - DS1/E1 Interface Type MIB;
- 1407 - DS3/E3 Interface Type MIB;
- 1414 - Identification MIB;
- 1418 - SNMP over OSI;
- 1419 - SNMP over AppleTalk;
- 1420 - SNMP over IPX;
- 1461 - Multiprotocol Interconnect over X.25 MIB;
- 1471 - PPP Link Control Protocol (LCP) MIB;
- 1472 - PPP Security Protocols MIB;
- 1473 - PPP IP Network Control Protocol MIB;
- 1474 - PPP Bridge Network Control Protocol MIB;
- 1512 - FDDI Interface Type (SMT 7.3) MIB;
- 1513 - Token Ring Extensions to RMON MIB;
- 1514 - Host Resources MIB;
- 1515 - IEEE 802.3 Medium Attachment Unit (MAU) MIB;
- 1525 - Source Routing Bridge MIB;
- 1565 - Network Services Monitoring MIB;
- 1566 - Mail Monitoring MIB;
- 1567 - X.500 Directory Monitoring MIB;
- 1573 - Evolution of the Interfaces Group of MIB-II;
- 1595 - SONET/SDH Interface Type MIB;
- 1604 - Frame Relay Service MIB;
- 1611 - DNS Server MIB;
- 1612 - DNS Server MIB;
- 1650 - Ether-Like Interface Type (SNMPv2);
- 1657 - BGP version 4 MIB;
- 1666 - SNA NAU MIB;
- 1694 - SMDS Interface Protocol (SIP) Interface Type MIB;
- 1695 - ATM MIB;
- 1696 - Modem MIB; and,
- 1697 - Relational Database Management System MIB.

Experimental:

- 1187 - Bulk table retrieval with the SNMP;
- 1224 - Techniques for managing asynchronously generated alerts;
- 1238 - CLNS MIB; and,
- 1592 - SNMP Distributed Program Interface (SNMP-DPI).

Informational:

- 1215 - A convention for defining traps for use with the SNMP;
- 1270 - SNMP communication services;
- 1303 - A convention for describing SNMP-based agents;
- 1321 - MD5 message-digest algorithm;
- 1470 - A network management tool catalog; and,
- 1503 - Automating Administration in SNMPv2 Managers.

Historical:

- 1156 - Management Information Base (MIB-I);
- 1161 - SNMP over OSI;
- 1227 - SNMP MUX protocol and MIB;
- 1228 - SNMP Distributed Program Interface (SNMP-DPI);
- 1229 - Extensions to the generic-interface MIB;
- 1230 - IEEE 802.4 Token Bus Interface Type MIB;
- 1232 - DS1 Interface Type MIB;
- 1233 - DS3 Interface Type MIB;
- 1252 - OSPF version 2 MIB;
- 1283 - SNMP over OSI;
- 1284 - Ether-Like Interface Type;
- 1286 - Bridge MIB;
- 1289 - DECnet phase IV MIB;
- 1298 - SNMP over IPX;
- 1304 - SMDS Interface Protocol (SIP) Interface Type MIB;
- 1316 - Character Device MIB;
- 1317 - RS-232 Interface Type MIB;
- 1318 - Parallel Printer Interface Type MIB;
- 1351 - SNMP Administrative Model;
- 1352 - SNMP Security Protocols;
- 1353 - SNMP Party MIB;
- 1368 - IEEE 802.3 Repeater MIB;
- 1398 - Ether-Like Interface Type MIB;
- 1596 - Frame Relay Service MIB;
- 1623 - Ether-Like Interface Type MIB;
- 1628 - Uninterruptable Power Supply MIB; and,
- 1665 - SNA NAU MIB.

Subscribing to SNMP-related Working Groups

Appletalk/IP Working Group:

- `apple-ip-request@cayman.com`

AToM MIB Working Group:

- `atommib-request@thumper.bellcore.com`

BGP Working Group:

- `iwg-request@ans.net`

Bridge MIB Working Group:

- `bridge-mib-request@nsl.dec.com`

Character MIB Working Group:

- `char-mib-request@decwrl.dec.com`

DECnet Phase IV MIB Working Group:

- `phiv-mib-request@jove.pa.dec.com`

FDDI MIB Working Group:

- `fddi-mib-request@cs.utk.edu`

Frame Relay Service MIB Working Group:

- `frftc-request@nsco.network.com`

Host Resources MIB Working Group:

- `hostmib-request@andrew.cmu.edu`

IEEE 802.3 Hub MIB Working Group:

- `hubmib-request@synoptics.com`

IDPR Working Group:

- `idpr-wg-request@bbn.com`

IDRP for IP Working Group:

- `idrp-for-ip-request@merit.edu`

Interfaces MIB Working Group:

- `if-mib-request@thumper.bellcore.com`

IPLPDN Working Group:

- `iplpdn-request@nri.reston.va.us`

IS-IS Working Group:

- `isis-request@merit.edu`

Mail and Directory Management Working Group:

- `ietf-madman-request@innosoft.com`

Modem Management Working Group:

- `modemmgt-request@telebit.com`

NOCtools Working Group:

- `noctools-request@merit.edu`

OSPF Working Group:

- `ospfigp-request@gated.cornell.edu`

PPP Working Group:

- `ietf-ppp-request@ucdavis.edu`

RIP Working Group:

- `ietf-rip-request@xylogics.com`

Remote Monitoring (RMON) Working Group:

- `rmonmib-request@jarthur.claremont.edu`

SNA DLC Services MIB Working Group:

- `snadlcmib-request@cisco.com`

SNA NAU Services MIB Working Group:

- `snanaumib-request@thumper.bellcore.com`

SNMPv2 Working Group:

- `snmp2-request@tis.com`

TCP Client Identity Protocol:

- `ident-request@nri.reston.va.us`

Trunk MIB Working Group:

- `trunk-mib-request@saffron.acc.com`

Uninterruptible Power Supply Working Group:

- `ups-mib-request@cs.utk.edu`

X.25 MIB Working Group:

- `x25mib-request@dg-rtp.dg.com`

Publication Information

The Simple Times is published with a lot of help from the SNMP community.

Publication Staff

Coordinating Editor:
Dr. Marshall T. Rose — Dover Beach Consulting, Inc.

Featured Columnists:
Dr. Jeffrey D. Case — SNMP Research, Inc.
University of Tennessee
Keith McCloghrie — Cisco Systems, Inc.
David T. Perkins — SynOptics Communications, Inc.
Steven L. Waldbusser — Carnegie Mellon University

Contact Information

Postal: ***The Simple Times***
c/o Dover Beach Consulting, Inc.
420 Whisman Court
Mountain View, CA 94043–2186
Fax: +1 415–968–2510
E-mail: st-editorial@simple-times.org
ISSN: 1060–6068

Submissions

The Simple Times solicits high-quality articles of technology and comment. Technical articles are refereed to ensure that the content is marketing-free. By definition, commentaries reflect opinion and, as such, are reviewed only to the extent required to ensure commonly-accepted publication norms.

The Simple Times also solicits terse announcements of products and services, publications, and events. These contributions are reviewed only to the extent required to ensure commonly-accepted publication norms.

Submissions are accepted only in electronic form. A submission consists of ASCII text. (Technical articles are also allowed to reference encapsulated PostScript figures.) Submissions may be sent to the contact address above, either via electronic mail or via magnetic media (using either 8-mm `tar` tape, $\frac{1}{4}$-in `tar` cartridge-tape, or $3\frac{1}{2}$-in MS-DOS floppy-diskette).

Each submission must include the author's full name, title, affiliation, postal and electronic mail addresses, telephone, and fax numbers. Note that by initiating this process, the submitting party agrees to place the contribution into the public domain.

Subscriptions

The Simple Times is available via electronic mail in three editions: *PostScript*, *MIME* (the multi-media 822 mail format), and *richtext* (a simple page description language). For more information, send a message to

```
st-subscriptions@simple-times.org
```

with a `Subject` line of

```
help
```

In addition, ***The Simple Times*** has numerous hard copy distribution outlets. Contact your favorite SNMP vendor and see if they carry it. If not, contact the publisher and ask for a list. (Communications via e-mail or fax are preferred).