September, 1993

SNMP MIB User,

The article *Understanding SNMP MIBs* which follows contains information and conventions that were state of the art as of the spring of 1992. Since then, the SNMPv2 working group was formed, did work, and concluded after producing a new management framework documented by RFCs 1441 through 1452. In doing this work, many of the open issues raised in *Understanding SNMP MIBs* have been resolved. Some can be applied retroactively to the SNMPv1 framework, but not all. It is my wish to update the current version of the article to add the resolutions, and expand on some of the advanced topics such as row creation/deletion and actions. However, at this time this work has not been scheduled.

I hope that you find the article informational and enjoyable. The most up-to-date information about the SNMP community can be found in the bi-monthly newsletter *The Simple Times*. To get information on subscribing to it, send a message to

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

with a `Subject` line of

```
help
```

And if you have any questions or comments about the article, please send me a message.

Enjoy,

Dave Perkins

Dave Perkins
SynOptics Communications, Inc
408 764-1516
dperkins@synoptics.com

Understanding SNMP MIBs

By David T. Perkins, September, 1993.

Revision 1.1.7

Understanding SNMP MIBs

By David T. Perkins, September, 1993
Revision 1.1.7

Table of Contents

Understanding SNMP MIBs

By David T. Perkins, September, 1993.

The first version of this article was published as "How to Write an SNMP MIB" in "Proceedings of the Nineteenth Internet Engineering Task Force." The second version was published as "How to Read and Use an SNMP MIB" in 3TECH, volume 1, number 4, Spring 1991. This third version has been corrected and updated to contain the latest information and practices as of the conclusion of the 23nd Internet Engineering Task Force (IETF) meeting in March 1992.

This article describes the definition of a management information base (MIB) used with the Simple Network Management Protocol (SNMP). It provides complete coverage starting with the formal definitions, then the unwritten conventions, and finally the current standard practices used in defining an SNMP MIB. With a properly designed MIB, SNMP can be used to manage network configuration, performance, faults, accounting, and security.

SNMP is becoming the standard, replacing past proprietary protocols for the management of network devices. The MIBs supported by these devices are the standard MIBs defined by the IETF working groups, as well as proprietary MIBs, called MIB extensions, defined typically by equipment or software vendors.

This article is written for designers of MIBs as well as the managers of network devices who must integrate the management tools with the management capabilities in devices. Readers should have a basic understanding of SNMP and the Open Systems Interconnection (OSI) abstract syntax notation one (ASN.1). The Simple Book[1] *by Marshall Rose is a good source of information on these topics but it is now, unfortunately, a little dated on MIBs. This article provides up-to-date date information that supersedes MIB information in* The Simple Book.

1. Introduction

This article explains how to read and use a management information base (MIB) defined for the simple network management protocol (SNMP). Managed devices are ones that can be monitored, controlled, and are capable of reporting events. With properly designed MIBs, SNMP can be used to manage network configuration, performance, faults, accounting, and security. A MIB defines managed objects using a framework called the structure of management information (SMI). The SMI defines how management information is grouped and named; allowed operations; permitted data types; and the syntax for specifying MIBs. The SMI, SNMP, and core MIBs were defined by working groups within the Internet Engineering Task Force (IETF).

Refinements to existing work, as well as the development of new MIBs are continuing within IETF working groups. MIBs developed outside the IETF by vendors of hardware or software are called MIB extensions or proprietary MIBs. MIB extensions are required for devices or software that have not yet had a MIB developed within the IETF. Also, they are needed to allow specific features of a proprietary system not included in a standard MIB to be managed. Much progress has been made in the last year.

There are MIBs being developed within the IETF for the most popular network device types and media types. See Figure 1 for a short categorization of MIBs with examples in each category.

Configuration	Interface Type	Protocol Stack	Functional
Hardware	Type Independent	DECnet IV	Router
System software	Token Ring	TCP/IP	Bridge
Firmware	Token Bus	OSI	Terminal Server
Trap Destinations	Ethernet (802.3)	XNS	Ethernet Repeater
Logging	DS1	IPX	Tokenring Repeater
Booting	DS3	SNA	Protocol Analyzer
Security	Async (serial)	AppleTalk	File Server
	Parallel		Print Server
	X.25		Name Server
	FrameRelay		Mail Server
	ISDN		Net management Agent
	FDDI		FDDI concentrator
	SMDS		

Figure 1 - MIB Categories and Examples

The IETF documents defining SNMP, SMI, and MIBs are cataloged in an on-line archival system. A document in this system is called a request for comments (RFC). These documents are easily accessed via a connection to the Internet. (See Figure 25 for instructions on how to get these documents.) The current status as well as the process of assigning status to standards is specified in RFC 1250[14]. Below in Figure 2 is a list of defining documents for SNMP, the SMI, and SNMP MIB formats. Surprisingly, these IETF documents are a poor source to consult for complete and consistent definitions. They are almost impossible, even for members of IETF working groups defining the MIBs, to read and apply. To be able to read, understand, and write MIBs has required learning the "verbal folklore" of SNMP. The intent of this article is to specify a complete and consistent definition of SNMP MIBs using the standards documents, the folklore, and implementation experience.

```
SNMP Protocol - RFC1157(Full): Case, J.D.; Fedor, M.; Schoffstall,
      M.L.; Davin, C.  Simple Network Management Protocol (SNMP).
      1990 May; 36 p. (Format: TXT=74894 bytes) (Obsoletes RFC
      1098)

SMI - RFC1155(Full): Rose, M.T.; McCloghrie, K.  Structure and
      identification of management information for TCP/IP-based
      internets.  1990 May; 22 p. (Format: TXT=40927 bytes)
      (Obsoletes RFC 1065)

Concise MIB format - RFC1212(Proposed): Rose, M.T.; McCloghrie,
      K.,eds.  Concise MIB definitions.  1991 March; 19 p.
      (Format: TXT=43579 bytes)

Trap format - RFC1215(Informational): Rose, M.T.,ed.  Convention
      for defining traps for use with SNMP.  1991 March; 9 p.
      (Format: TXT=19336 bytes)
```

Figure 2 - SNMP Defining Documents

2. Notations for Syntax Descriptions

The notational rules for the syntax specifications in this article are shown below:

- Literal values are specified in quotes, for example **"read-only"**;
- Replaceable items are surrounded by less than and greater than characters, for example **<oidItem>**;
- Ellipsis are used to indicate that the previous item may be repeated one or more times, for example **<smiItem>...**;
- Square brackets are used to enclose optional items, for example **["DEFVAL" "{" <defaultValue> "}"]**;
- Curly braces are used to group together items, for example **{ "OBJECT" "IDENTIFIER" }**;
- A vertical bar is used to indicate a choice between items, for example **<oidItem> | <objectItem> | <seqItem> | <trapItem>**;
- The equals character is used to mean "defined as"; for example **<mib> = <module>...** .

3. What is a MIB?

The term MIB has different meanings based on its context. Generally, a MIB describes information that can be obtained and/or modified via a network management protocol. This information enables systems on a network to be managed.

The OSI community divides network management into five functional areas:

- Configuration Management: names all elements in a network and specifies their characteristics and state. This includes the information needed so maps of the network can be drawn, and exploded diagrams of device components can be shown by network management station application programs.
- Performance Management: determines the effective utilization of the network and components of network devices. Performance analysis helps network managers monitor network availability, response time, throughput, and resource usage.
- Fault Management: detects, isolates, and corrects network problems.
- Security Management: controls access and protection of information on the network from disclosure or modification.
- Accounting: measures usage and computes costs based on specified policies. This information can be used for billing applications.

Managed devices are ones that can be monitored and controlled and are capable of reporting events. The OSI management protocol called Common Management Information Protocol (CMIP) includes the following operations:

get	retrieves specified information;
set	changes the value of specified information;
action	performs an imperative command, such as reset an interface;
create	forms a new instance of a managed object;
delete	removes a specified object instance;
event-report	signals to a manager that an event of importance has occurred.

SNMP includes the following operations:

get	same function as OSI *get;*
getnext	used for table row retrieval and for discovery of managed objects;
set	same function as OSI *set;*
trap	same function as OSI *event-report.*

The OSI *action*, *create*, and *delete* operations have no direct mapping to SNMP operations. However, the functionality of these operations can be implemented with SNMP *get* and *set* operations and proper design of SNMP MIB variables.

The OSI and SNMP models of a MIB are very different. At the time that the SMI[3] and MIB-I[4] RFCs were written, it was thought that one MIB could be designed for both management protocols. This approach was outlined in RFC1052[8]. Time has shown that while it is possible to constrain an OSI MIB so it can be mapped to an SNMP MIB, it is not possible to mechanically map arbitrary MIBs between the two management protocols. RFC1109[9] noted the difficulties in a dual approach and allowed SNMP and CMIP over TCP/IP (CMOT) to proceed independently. Some of the difficulties with the SMI and SNMP definition documents can be traced directly back to the original approach as specified in RFC1052[8]. Reading these RFCs (and reading in between the lines) is an interesting education in SNMP history, but they don't help define MIBs for SNMP.

4. The Structure of Management Information (SMI)

As already described, a MIB defines managed objects using a framework called the SMI. The SMI defines how management information is grouped and named; allowed operations; permitted data types; and the syntax for specifying MIBs. Managed objects are abstractions of resources on systems that exist independently of their need to be managed. The MIB for these objects does not define the actual realization of these resources. Some objects (such as the location of a system) have only one instance, while others (such as network connections) have multiple instances. Related objects that have the same type of instance are organized into conceptual tables in SNMP MIBs. The identity of an object together with its associated instance is called an SNMP variable.

The SMI is similar to the schema for a database system. It defines the model of managed objects and the operations that can be performed on the objects, as well as data types that are permitted for the objects. The OSI approach is similar to an object-oriented model, while the SNMP approach is similar to a relational database model.

4.1. Object Identifiers (OIDs)

Objects are unambiguously identified (or named) in SNMP by assigning them an object identifier (OID). Globally unique for all space and time, OIDs are a sequence of nonnegative integers organized hierarchically like UNIX or PC-DOS file system names. For ease of use, a textual name is associated with each sequence element, or component, of an OID. The last component name is used by itself as a shorthand way of naming an object. To help make sure there is no confusion in naming an object since the shorthand names do not have to be unique, all textual names of objects defined by IETF working groups are, by convention, made unique by using a different prefix for objects in each new MIB. SNMP uses an encoded form of the numeric value, not the textual name. OIDs are written in one of the following formats:

```
Syntax:
      "{"  { {<name>["("<number>")"]}  |  <number>}...  "}"
      or
      <number> ["."<number>]...

Where
      <name> is a component name; and
      <number> is a component value.
```

For example:

```
{ iso org(3) dod(6) internet(1) }  or  1.3.6.1

{ internet 4 }  or  1.3.6.1.4

{ tcp 4 }  or  1.3.6.1.2.1.6.4
```

The precise syntax to specify an OID depends on where the OID is used. These examples illustrate a shorthand specification of OIDs. The first <name> in the OID can be any name in the hierarchical OID tree as long as it is unique within the context that it is used. There is no shorthand for specifying OIDs in the numeric form. The use of <name>"("<number>")", by convention, is not used in IETF developed MIBs.

OIDs can be used to uniquely identify anything, not just managed objects. Some OIDs are used just as placeholders to help organize the OID hierarchy (see Figure 3).

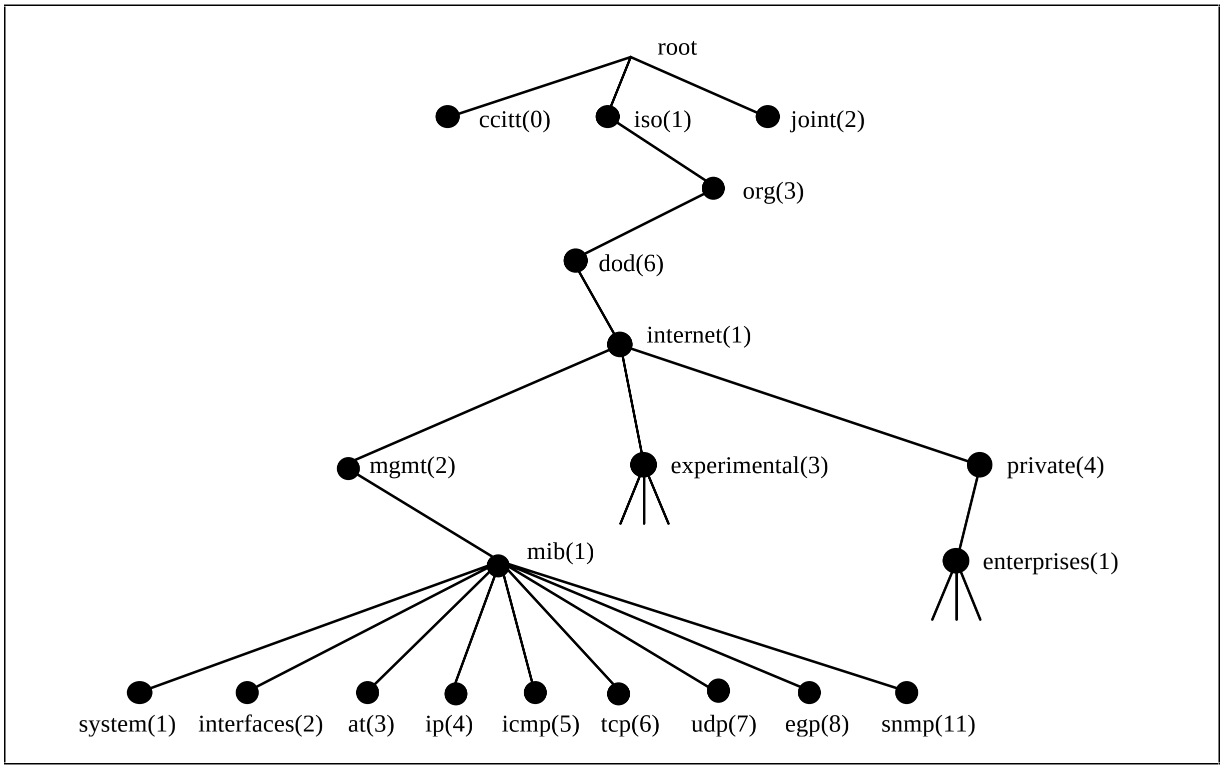

Figure 3 - OID Tree for SNMP

The following OID prefixes that are important when writing an SNMP MIB:

internet	which is defined as { iso(1) org(3) dod(6) 1 }
mgmt	which is defined as { internet 2 }
experimental	which is defined as { internet 3 }
private	which is defined as { internet 4 }
mib, mib-1, and mib-2	which are defined as { mgmt 1 }
enterprises	which is defined as { private 1 }

Objects for standard SNMP MIBs are defined under the "mib" branch of the hierarchy. The existence of "mib-1" and "mib-2" are the result of a versioning scheme that didn't quite work out. Present day MIBs should reference "mib-2". Experimental MIBs being developed by IETF working groups define objects under the "experimental" branch. Proprietary MIBs define objects within an organization's subtree located under the "enterprises" branch. To get a number under the enterprises branch, simply contact the Internet Assigned Numbers Authority (IANA) and request an enterprise number. The assignment of numbers within an enterprise is determined locally. IETF working groups should obtain a number under the experimental branch through coordination with the network management area director and the IANA. (See Figure 26 for contact information to the internet assigned numbers authority.)

Note 1: the SMI RFC[3] reserves for "later use" (i.e., for yet to be specified reasons) the component value 0 for items defined within the "mib" subtree. Items defined in other subtrees such as under "enterprises" are not restricted.

Note 2: the SMI RFC[3] requires that all item names defined within the "mib" subtree be unique, mnemonic, and a printable string. Items defined in other subtrees such as under "enterprises" do have this requirement.

4.2. MIB Modules

The MIB for SNMP is defined by a collection of module definitions that may be contained in one or more documents. Some people view the union of all management information to be "the MIB", since to them there is only one MIB even though it is made up of a collection of documents. However, most people use the term MIB to mean one or more modules that contain definitions of related management information. MIB-I and MIB-II are the first and second generations of the core definitions for TCP/IP hosts and routers. Additional MIBs have been defined for routing protocols such as OSPF and BGP, interface types such as token ring and DS1, protocol stacks such as DECnet, and device types such as bridges and terminal servers. Proprietary MIBs have been defined to extend existing MIBs or to specify areas not covered by IETF-authored MIBs.

Modules are a mechanism to scope names. For all MIBs that are intended to become IETF standards, the SMI RFC[3] requires names to be unique across all MIB modules. This is not a requirement for experimental or proprietary MIBs. It would be difficult, if not impossible, to enforce this "rule" across all proprietary MIBs which are usually developed in secrecy and integrated by the end-user (who will be the first to discover "duplicate" names). As long as the module names are unique and all object names within a module unique, then it is allowable to have duplicate object names in separate modules. (Using duplicate names should be avoided where possible.)

MIB module names must start with an uppercase letter that is followed by an arbitrary number of letters, digits, and hyphens. A hyphen can not be the last character nor can a hyphen be immediately followed by another hyphen. However, MIB compilers or network management station software may put length restrictions on names.

Within a module are defined registration points in the OID tree, SNMP managed objects, values for items that have syntax type of object identifier, traps, sequences, and textual conventions. Each of these will be described in detail later in this article. Shown in Figure 4 is the syntax for SNMP MIBs and MIB modules.

```
Syntax:
      <mib> = <module>...

      <module> =
            <ModName> [ <modId> ] "DEFINITIONS" "::=" "BEGIN"
               [ "IMPORTS"  <importList>... ";" ]
               [ <smiItem>... ]
               [ <textConvItem>... ]
               { <oidItem> | <objectItem> |
                 <seqItem> | <trapItem> }...
            "END"

      <importList> = <importItem> [ "," <importItem> ]...
                        "FROM" <ImportModName>

Where
      <ModName> is the name of a MIB module;
      <modId> is a now obsolete method to assign an object
            identifier to a module;
      <smiItem> is a definition for SMI items such as syntax
            types, the OBJECT-TYPE and TRAP-TYPE macros;
      <importItem> is an item defined in another MIB module;
      <ImportModName> is the name of another previously defined
            MIB module;
      <textConvItem> is a definition of a textual convention;
      <oidItem> is a definition of an object identifier;
      <objectItem> is a definition of an item with the
            OBJECT-TYPE macro;
      <seqItem> is a definition of a sequence; and
      <trapItem> is a definition of an item with the
            TRAP-TYPE macro.
```

Figure 4 - Syntax for SNMP MIBs and MIB Modules

The use of <modId> for modules, the result of a versioning scheme that didn't quite work out, is now obsolete. Older proprietary MIBs may have this syntax. A new MIB may not define any additional SMI items such as new syntax types or ASN.1 macros. The IMPORTS clause allows items defined in other modules to be used in the module being defined. This is useful (and necessary) to pick up the definitions of the OBJECT-TYPE macro from the Concise MIB RFC[6] or the syntax types defined in the SMI RFC[3]. An item may only be imported if the module has been previously defined. Textual conventions are a "conforming" mechanism to extend the SMI syntax types without adding a new syntax type encoding. Object identifier items are registration points in the OID tree, values for items that have the syntax of object identifier, and groups for SNMP objects. The OBJECT-TYPE macro is used to define tables, rows, and simple and columnar objects. Sequences are used in defining "conceptual" tables. Lastly, the TRAP-TYPE macro is used to define traps. Within a MIB definition, ASN.1 comments may be used to provide additional insight into the organization and definition of items within the MIB. Comments begin with two hyphens and end with the end of the line.

Note: a reference citation in a comment should write out the name of the reference document and not use a footnote or "[number]" notation. Existing MIBs that use the "[number]" notation are being modified. Typically a MIB is "extracted" from its defining document to be used in a managment station. After this is done, all reference citations that use a footnote notation are useless since the reference section will not be included with the "extracted" MIB.

4.3. Mechanics of Module Specification

The position in the OID tree of the objects defined in the module must be determined before writing a MIB module. For MIBs developed by IETF working groups, a branch should be assigned under the "internet experimental" branch. For private MIBs, a branch needs to be assigned under an "enterprises" branch in the "internet private" tree. Local customs determine the scheme used for assignments under each "enterprises" branch. One local custom is to create both an "experimental" branch and a branch for each released MIB module. Branches can be created within the experimental branch for testing and prototyping. Once a MIB is published, items cannot be changed; they can only be obsoleted and re-created. Thus, the standard practice is to design a MIB under an "experimental" branch, test it out, and then move it to a "standard" branch when the MIB document is published.

MIB modules almost always start with the same format. A unique name is chosen for the module followed by imports for the registration point objects; for syntax types used within the module; and for the OBJECT-TYPE and TRAP-TYPE macros. Items should be imported only when used within a module. See Figure 5 for an example of a shell for a MIB module.

```
Example:
      EXAMPLE-MIB DEFINITIONS ::= BEGIN

      -- Copyright notice

      -- MIB Descriptions

      -- Some or all the the following IMPORTS...
      IMPORTS
            enterprises, Counter, Gauge,
            TimeTicks, IpAddress          FROM RFC1155-SMI
            DisplayString, PhyAddress     FROM RFC1213-MIB
            OBJECT-TYPE                   FROM RFC-1212
            TRAP-TYPE                     FROM RFC-1515;

      -- Some or all of the following:

      -- Textual Conventions

      -- Registration points

      -- Groups

      -- SNMP managed Objects

      -- SNMP traps

      END
```

Figure 5 - SNMP MIB Module

4.4. Object Identifier Items

In SNMP MIBs, items defined as object identifiers are high-level registration points in the object identifier tree, groups, or values for items that have the syntax type of object identifier. Names of object identifier items must start with a lowercase letter that is followed by an arbitrary number of letters, digits, and hyphens. A hyphen can not be the last character nor can a hyphen be immediately followed by another hyphen. However, MIB compilers or network management station software may put length restrictions on names.

Items "mib", "experimental", and "enterprises" as shown in Figure 3 are simply placeholders in the OID tree. A special type of placeholder is an SNMP group is used to organize MIB objects together. For example, in MIB-II[5], "system" is a group containing objects specifying descriptive, high-level information about a managed device such as location. Besides organizing together similar objects, groups are also used to specify units of conformance. Groups can be either mandatory or optional. When they are mandatory, all contained objects must be present for a device to claim conformance to the MIB defining the group. When they are optional, the entire set of contained objects must be present when the capability described by the objects is present, and all contained objects should be absent when the capability described by the objects is not present. For example, if a managed device is a router that implements the EGP routing protocol, then it must implement the EGP group to claim conformance to MIB-II.

Finally, items may be defined as object identifiers so that they may be returned as the value of an item that has the syntax type of object identifier. For example, the item sysObjectID which is defined in the MIB-II RFC[5] has the syntax type of object identifier. The value of this item specifies the type of device that is being managed. Each vendor should define object identifier items for each type of SNMP-managed device it produces. See Figure 6 for the syntax of object identifier definitions and an example. ASN.1 comments should be used to fully describe OID items. A nice addition to the next version of the Concise MIB RFC[6] would be an "OID" macro so the description of the item could be specified using a DESCRIPTION clause and the status could be specified using a STATUS clause.

```
Syntax:
      <oidItem> = <oidName> "OBJECT" "IDENTIFIER" "::="
                        "{" <parent> <number> "}"

Where:
      <oidName> is the name of the OID item being defined;
      <parent> is the name of the directly containing item
            in the OID tree; and
      <number> is the value of the last component of the
            item being defined.

Example:
      egp OBJECT IDENTIFIER ::= { mib-2 8 }
```

Figure 6 - Syntax and Example for Object Identifier Items

4.5. Managed Object Definitions

Three types of objects that can be defined using the OBJECT-TYPE macro: tables, rows, and leaf objects (simple and column). The names of these items must start with a lowercase letter that is followed by an arbitrary number of letters, digits, and hyphens. A hyphen can not be the last character nor can a hyphen be immediately followed by another hyphen. However, MIB compilers or network management station software may put length restrictions on names.

There are two versions of the ASN.1 OBJECT-TYPE macro. The earlier version is defined in the SMI RFC[3] and has been replaced by the version defined in the Concise MIB RFC[6]. Older MIB documents that have not been updated will use the now obsolete version of the OBJECT-TYPE macro. The most important difference between the two versions is that the current version contains the INDEX clause that is used to specify (indirectly) the rules to construct the instance portion of SNMP variables. Shown in Figure 7 is the simplified version of the syntax for the OBJECT-TYPE macro:

```
Syntax:
      <objectItem> = <objectName> "OBJECT-TYPE"
                  "SYNTAX" <syntax>
                  "ACCESS" <access>
                  "STATUS" <status>
                  [ "DESCRIPTION" <description> ]
                  [ "REFERENCE" <reference> ]
                  [ "INDEX" "{" <indexItems> "}" ]
                  [ "DEFVAL" "{" <defaultValue> "}" ]
                  "::=" "{" <parent> <number> "}"

      <objectName> = <tableName> | <rowName> | <leafName>

      <syntax> =  { "SEQUENCE" "OF" <SequenceName> } |
                  <SequenceName> |
                  <leafSyntax>

Where:
      <objectName> is the name of the item being defined;
      <parent> is the name of the directly containing item
            in the OID tree;
      <number> is the value of the last component of the
            item being defined; and
      Values for <access>, <status>, <leafSyntax>, etc. are
            defined later.
```

Figure 7 - Simple Syntax for OBJECT-TYPE Macro

Following are descriptions for each type of object. The OBJECT-TYPE macro is limited to the clauses that may be used for each type of object. Following the object descriptions are detailed descriptions for each of the individual clauses in the OBJECT-TYPE macro.

4.5.1. Table Objects

A table consists of rows. Table construction is somewhat strange due to the history of the definition of the SNMP SMI. A table is not retrievable via SNMP; only the "columnar" objects within a table can be retrieved. The syntax for a table must be "SEQUENCE OF <sequence>". By convention, tables are named with the suffix "Table". By convention also, the name of the associated sequence is the prefix of the table name (without the "Table" suffix) and the first letter is changed to uppercase. The suffix of the sequence, by convention, is set to "Entry". For example, if "fooTable" was the name of a table, then the associated sequence would be named "FooEntry". The access for tables must be "not-accessible". See Figure 9 for syntax of table object definitions and an example.

```
Syntax:
      <tableName> "OBJECT-TYPE"
            "SYNTAX" "SEQUENCE" "OF" <SequenceName>
            "ACCESS" "not-accessible"
            "STATUS" <status>
            [ "DESCRIPTION" <description> ]
            [ "REFERENCE" <reference> ]
            "::=" "{" <parent> <number> "}"

      <tableName> = <name>"Table"
      <SequenceName> = <Name>"Entry"

Where:
      <name> is the prefix of the table being defined;
      <Name> is the prefix of the associated sequence;
      <parent> is the name of the directly containing item
            in the OID tree;
      <number> is the value of the last component of the
            item being defined; and
      Values for <status>, <description>, etc. are
            defined later.

Example:
      ifTable OBJECT-TYPE
            SYNTAX SEQUENCE OF IfEntry
            ACCESS not-accessible
            STATUS mandatory
            ::= { interfaces 2 }
```

Figure 9 - Syntax and Example for Tables

4.5.2. Row Objects

A row consists of columns. A row is not retrievable via SNMP; only the "columnar" objects within a row can be retrieved. A *get* or *getnext* request can be used to get all the columns for a selected row. The name of a row, by convention, is taken as the name of the associated table with the "Table" suffix replaced by "Entry". The syntax type for a row must be the sequence used by the associated table. The access for the row must be "not-accessible". The OID value for the row must be the same as the associated table with the addition of a single component of value 1. The INDEX clause must be used to specify rules for "instance" construction of the columnar objects of the table ("instances" are defined under "Leaf Objects" and in greater detail in "Considerations for Instances"). See Figure 10 for the syntax of row object definitions and an example.

```
Syntax:
      <rowName> "OBJECT-TYPE"
            "SYNTAX" <SequenceName>
            "ACCESS" "not-accessible"
            "STATUS" <status>
            [ "DESCRIPTION" <description> ]
            [ "REFERENCE" <reference> ]
            "INDEX" "{" <indexItems> "}"
            "::=" "{" <tableName> 1 "}"

<rowName> = <name>"Entry"
<SequenceName> = <Name>"Entry"
<tableName> = <name>"Table"
<indexItems> = <indexItem> [ "," <indexItem> ]...

Where:
      <name> is the prefix of the row being defined and
            the prefix of the associated table;
      <Name> is the prefix of the associated sequence;
      <indexItem> is the name of an item in the sequence
            for the row (or a syntax type name); and
      Values for <status>, <description>, etc are
            defined later.

Example:
      ifEntry OBJECT-TYPE
            SYNTAX IfEntry
            ACCESS not-accessible
            STATUS mandatory
            INDEX { ifIndex }
            ::= { ifTable 1 }
```

Figure 10 - Syntax and Example for Rows

4.5.3. Sequence Definitions

A sequence is used to specify the "columns" in a row. The name of a sequence must start with an uppercase letter that is followed by an arbitrary number of letters, digits, and hyphens. A hyphen can not be the last character nor can a hyphen be immediately followed by another hyphen. However, MIB compilers or network management station software may put length restrictions on names.

For most tables and rows the items of the sequence are the "children" of the row object. However, if the table is an extension to an existing table, all or a subset of the columnar objects from the existing table may additionally be added to the sequence. There is some controversy over the exact use of sequences and thus which items should be included. (Unfortunately, the Concise MIB RFC[6] did not clarify or eliminate the awkward and error-prone syntax for specifying tables, rows, and sequences by adding a "table" macro.) In addition to the names of the columnar items for the row, the syntax for these items must be specified. Instead of a word-for-word match of the item's syntax, a simplified version should be given. The size and range clauses, if used in the definition of the item, do not have to be specified. If they are, the values must match the definition. The values for enumerated integers may not be specified. The name of the sequence should, by convention, be the same as the row name with the first letter changed to uppercase. A sequence can be used with only one table and row. Also, a sequence may not be imported from another module. See Figure 11 for the syntax of sequence definitions and an example.

```
Syntax:
      <seqItem> = <SequenceName> "::=" "SEQUENCE" "{"
                        <columnItem> <leafSyntax>
                        { "," <columnItem> <leafSyntax> }...
                  "}"

<SyntaxName> = <Name>"Entry"

Where:
      <Name> is the prefix of the sequence being defined;
      <columnItem> is name on an item in the sequence; and
      <leafSyntax> is the simplified syntax for the item.

Example:
      ipAddrEntry ::= SEQUENCE {
            ipAdEntAddr       IpAddress,
            ipAdEntIfIndex    INTEGER,
            ipAdEntNetMask    IpAddress,
            ipAdEntBcastAddr  INTEGER,
            ipAdEntReasmMaxSize INTEGER
            }
```

Figure 11 - Syntax and Example for Sequences

4.5.4. Leaf Objects

A leaf object is the smallest grouping of information. Together, an object's identity and associated "instance" locate one piece of information, called an SNMP variable. These are the operands in SNMP operations. "Simple" leaf objects such as the number of interfaces in a device, have only a single instance. In this case, the instance is specified as one component with a value of zero. "Columnar" leaf objects are organized into conceptual tables. This is done because it is possible to have none, one, or multiple instances of the information, such as TCP connections to a device. The instance format for a columnar object is determined by the value of the INDEX clause for the containing row. Simple objects are organized under a group and are usually given the same prefix as the group. Columnar objects are organized under a row and are usually given the same prefix as the row. See Figure 12 for syntax of leaf object definitions and examples. (The section called "Considerations for Instances" contains details about "instance" encodings.)

```
Syntax:
      <leafName> "OBJECT-TYPE"
            "SYNTAX" <leafSyntax>
            "ACCESS" <access>
            "STATUS" <status>
            [ "DESCRIPTION" <description> ]
            [ "REFERENCE" <reference> ]
            [ "DEFVAL" "{" <defaultValue> "}" ]
            "::=" "{" <parent> <number> "}"
Where:
      <leafName> is the name of the leaf object being defined;
      <parent> is the name of the directly containing item
            in the OID tree (a row or a group);
      <number> is the value of the last component of the
            item being defined; and
      Values for <leafSyntax>, <access>, <status>, etc.  are
            defined later.

Examples:
      sysUpTime OBJECT-TYPE
            SYNTAX TimeTicks
            ACCESS read-only
            STATUS mandatory
            ::= { system 2 }

      ipAdEntAddr OBJECT-TYPE
            SYNTAX IpAddress
            ACCESS read-only
            STATUS mandatory
            ::= { ipAddrEntry 1 }
```

Figure 12 - Syntax and Examples for Leaf Objects

4.5.5. Textual Conventions

A textual convention is an item used to specify additional semantics to an existing syntax type. This shorthand mechanism was invented in the MIB-II RFC[5] and is the only allowable method to "extend" the SMI syntax types in SNMP. The two textual conventions defined in MIB-II are DisplayString and PhysAddress. The SMI base syntax type for both is OCTET STRING. Thus, no new encoding is transmitted in SNMP PDUs when these types are used. However, items defined as having the syntax of DisplayString have their OCTET values restricted to printable ASCII characters. Textual conventions are a new and powerful technique that is just beginning to be widely used in defining MIBs. Network management station software has not yet caught up with this technique and most, if not all, does not correctly process MIBs that use textual conventions. For some, the current support is limited to recognizing DisplayString and PhysAddress. Hopefully, by the summer of 1992 most management station software will be upgraded to support textual conventions. See Figure 13 for the syntax of textual convention definitions and an examples .

```
Syntax:
      <textConvItem> = <textConvName> "::=" <leafSyntax>

Where:
      <textConvName> is the name of the textual convention
            being defined; and
      <leafSyntax> is any defined syntax type including a
            textual convention.

Examples:
      DisplayString ::= OCTET STRING
      Status ::= INTEGER  { enabled(1), disabled(2) }
```

Figure 13 - Syntax and Examples for Textual Conventions

A comment describing the semantics of each textual conventions should be given in the MIB. A nice addition to the next version of the Concise MIB RFC[6] would be a "textual convention" macro so that the description of the textual convention could be specified using a DESCRIPTION clause. A size or range may be specified when the textual convention is defined or used. If so, these clauses must be valid with the base syntax type and cannot be specified when they are both defined and used. A textual convention name must start with an uppercase letter that is followed by an arbitrary number of letters, digits, and hyphens. A hyphen can not be the last character nor can a hyphen be immediately followed by another hyphen. However, MIB compilers or network management station software may put length restrictions on names.

4.5.6. Values for SYNTAX

The value for <syntax> determines the type of the managed object. As specified earlier, table objects have a value of "SEQUENCE OF <SequenceName>". Row objects have a value of "<SequenceName>". Note that the first character of the sequence's name is uppercase. Leaf objects can not specify the name of a sequence for their syntax. Their value for <syntax> is defined by <leafSyntax> as shown in Figure 14.

```
Syntax:
      <leafSyntax> = { "INTEGER" [ <range> | <enums> ] } |
                  { "OCTET" "STRING" [ <size> ] } |
                  { "OBJECT" "IDENTIFIER" } |
                  { <smiApplType> } |
                  { <textConvName> [ <range> | <size> ] }

      <smiApplType> = "NetworkAddress" | "IpAddress" |
                  "Counter" | "Gauge" | "TimeTicks" |
                  "Opaque"

      <range> = "(" <lower> ".." <higher> ")"
      <enums> = "{" <enumItem> [ "," <enumItem> ]... "}"
      <enumItem> = <enumName> "(" <enumValue> ")"
      <size> = "(" "SIZE" "(" <smallest> [ ".." <largest>] ")" ")"

Where:
      <lower> and <higher> may be positive or negative
            integers, bitstring constants, or hexstring constants;
      <enumName> starts with a lowercase letter followed
            by arbitrary number of letters, digits, and hyphens.
            A hyphen can not be the last character nor can a
            hyphen be immediately followed by another hyphen;
      <enumValue> is a positive integer, bitstring, or
            hexstring constant not having a value of zero; and
      <smallest> and <largest> may be non-negative
            integers, bitstring constants, or hexstring constants.
```

Figure 14 - Syntax for "Leaf Syntax"

The following describes the details for each syntax type:

INTEGER Integers which may have an associated value range assigned to them. Integers by convention must fit into 32 bits. (Note: The range of the integer should be, but is not always, specified. This indicates the implementation characteristics to both the agent and manager writers.) Newer MIBs are starting to use hex constants to specify the range when doing so makes the range more obvious. If so, these values are assumed to be positive numbers.

```
examples:
      SYNTAX INTEGER (0..65535)
      SYNTAX INTEGER (0..'ffff'h)
      SYNTAX INTEGER (0..'ff'H)
```

<enumerated>

A special case of integer. Zero and negative numbers are not permitted values according to the SMI RFC[3]. However, proprietary MIBs may incorrectly use these values. The object must take only those values that are listed in the enumeration. A value called "other" should be provided, but is not always present in older MIBs. The object's DESCRIPTION clause should describe the values that are not obvious.

```
examples:
      SYNTAX INTEGER {
            gateway(1),
            host(2) }
      SYNTAX INTEGER {
            other(1),
            invalid(2),
            direct(3),
            indirect(4) }
```

<integerBitstring>

A special case of the integer. (Note: This is also called a "sum".) This is usually used for short bit strings that are 32 or fewer bits long. Bit numbering starts at zero, which is positioned at the low order end of the integer. The DESCRIPTION clause should specify the position (i.e., value) of each bit. Descriptions of the bit values are typically written using 2 raised to the power of the bit number.

```
example:
      SYNTAX INTEGER (0..127)
```

OCTET STRING

A string of bytes that may have an associated size assigned to them. (Note: The size of the string should be, but is not always, specified. This indicates the implementation characteristics to both the agent and manager writers.) Items defined as an octet string are intended to be binary bytes of data. The size is fixed if only one "size" is specified and is varying if no "size" or two "sizes" are specified.

examples:

```
      SYNTAX OCTET STRING (SIZE (0..9))  -- varying
      SYNTAX OCTET STRING                -- varying
      SYNTAX OCTET STRING (SIZE (6))     -- fixed
```

<octetBitstring>

A special case of the octet string type. This is usually used for long bit strings (i.e. those longer than 32 bits). Bits are numbered starting with zero, which is positioned as the high order bit of the first byte. Bits are packed eight to a byte. Any unused bits in the last byte should be set to zero, but must be ignored. The DESCRIPTION clause should describe each bit, with the size specified as a constant number of octets. There may be some applications where the size may be variable.

example:

```
      SYNTAX OCTET STRING (SIZE (4))
```

DisplayString	A well known textual convention which is a special case of the octet string type where all the bytes are printable ASCII characters. Sometimes, incorrectly, a variable may include formatting characters such as CR and LF, and the C programming language string terminator character zero. (Note: The size of the String should be, but is not always, specified. This indicates the implementation characteristics to both the agent and manager writers.) example: `SYNTAX DisplayString (SIZE (0..256))`
PhysAddress	A well known textual convention which is a special case of the octet string type where all the bytes are the binary in "network order" of a media- or physical-level address. (Note: The size of the string should be, but is not always, specified. This indicates the implementation characteristics to both the agent and manager writers.) example: `SYNTAX PhysAddress (SIZE (6))`
OBJECT IDENTIFIER	An object identifier value. example: `SYNTAX OBJECT IDENTIFIER`
<ObjectName>	Special case of the object identifier, where the value is restricted to the OIDs of MIB objects and trees.
NetworkAddress	Used to indicate an address choice from one of the possible protocol families. Currently, only IP addresses are supported. Use of this type is being discontinued.
IpAddress	A four byte octet string in network order.
Counter	A nonnegative integer that counts up to 2^{32}-1 and wraps back to zero. A range may not be specified.
Gauge	A nonnegative integer that may increase or decrease but which latches at its maximum value of 2^{32}-1. A range may not be specified.
TimeTick	A nonnegative integer that counts time in hundredths of seconds since some epoch with a limit of 2^{32}-1. (The description of its use must identify the reference epoch.) A range may not be specified.
Opaque	A data type to encapsulate an arbitrary ASN.1 encoded data item. This is usually used to hold data types for private MIBs that are not a type defined above. This results in the original data being double-wrapped. (Note: Use of this type is strongly discouraged.)

4.5.7. Values for ACCESS

The values for the ACCESS clause are specified as "read-only", "read-write", "write-only", and "not-accessible" in the SMI RFC[3]. However, in SNMP MIBs, tables and rows must have the value of "not-accessible" and leaf objects must have either "read-only" or "read-write" values. An SNMP "MIB-view" may add additional restrictions (or capabilities) to leaf objects so that <access> can be changed from "read-write" to "read-only" or "not-accessible". Each object in a conformant MIB must have at least one "MIB-view" that implements the <access> one-for-one with the MIB specification.

4.5.8. Values for STATUS

The values for the STATUS clause are specified as "mandatory", "optional", "obsolete", and "deprecated" by the SMI RFC[3]. However, in SNMP MIBs, "optional" is not allowed. A whole subsection (group) may be optional, but individual objects can not be labeled as "optional". An implementation of an agent may not have access to the value of a "mandatory" object. In this case, *gets* and *sets* to the object should return "noSuchName" errors and *getnexts* should simply return the next lexicographically ordered object. Unimplemented objects in a mandatory group result in the agent being labeled as "non-conformant" to that MIB group. An agent that returns "benign" values for readable objects or does not change writable objects is also labeled as "non-conformant". The "obsolete" value is meant to document the existence of an object that is no longer supported. However, the current practice is to remove obsolete objects from a MIB and leave only a comment to indicate that they are no longer used. Objects for which support will soon be dropped are given the value "deprecated". These objects are mandatory until they are made "obsolete".

4.5.9. Values for DESCRIPTION

The value for the optional DESCRIPTION clause as defined in the Concise MIB RFC[6] is a "textual definition of that object type which provides all semantic definitions necessary for implementation". It is meant as information for agent and manager writers and not as the "help text" for manager users. A reference citation in the text should write out the name of the reference document and not use a footnote or "[number]" notation. Existing MIBs that use the "[number]" are being modified. Typically a MIB is "extracted" from its defining document to be used in a management station. After this is done, all reference citations that use a footnote notation are useless since the reference section will not be included with the "extracted" MIB. The actual value for the DESCRIPTION clause is a string enclosed in double quotes that may be continued across multiple lines. It may not contain double quotes. It is customary to line up continuation lines of a multiline quote in the same starting column and leave the right edge ragged with no hyphenation. If quotes are needed within the quoted text, then single quotes (ASCII apostrophes) are used.

4.5.10. Values for REFERENCE

The value for the REFERENCE clause is a "textual cross-reference" to another document that describes the same object. This is used when converting MIBs from other formats such as IEEE or OSI to SNMP MIBs. A reference citation in the text should write out the name of the reference document and not use a footnote or "[number]" notation. Existing MIBs that use the "[number]" notation are being modified. The actual value is a quoted text string that follows the syntax rules used for description values.

4.5.11. Values for INDEX

The INDEX clause must be included for row objects, and no other types of objects. This is the current standard practice. It lists the columns of the row that are used as the instance specifiers. The order of the items in the INDEX clause indicates the order that instance components are used in constructing the instance information for columnar objects in the containing table. (See "Considerations for Instances", below, for more information on how this clause is used.) For backward compatibility with older MIBs, the Concise MIB RFC[6] allows the use of the names of syntax types instead of items in the sequence that defines the row. The syntax types are limited to the following: INTEGER, OCTET STRING, OBJECT IDENTIFIER, NetworkAddress, and IpAddress. Use of a syntax type name is strongly discouraged. None of the IETF developed MIBs use this construct. When used, the DESCRIPTION clause must specify the semantics associated with <indexItems>. Another use of the INDEX clause, typically found in MIBs developed before the Concise MIB RFC was written, is with leaf objects. The authors of these MIBs didn't use the familiar construct of defining a table and a row to contain the columnar objects. When these MIBs were converted to the concise format, the INDEX clause was added to the leaf objects since no row objects existed. This "liberal interpretation" of the concise MIB rules has caused quite a few problems (since most consider it to be illegal).

4.5.12. Values for DEFVAL

The DEFVAL clause may only be included for leaf objects that are columns in tables that are not used as the instance items. The value should be used as a hint for the agent implementation when a row is created and the *set* request creating the row does not contain values for all the writable columns in the row. See Figure 15 for a list of syntax values and examples. NOTE: Hex strings and bit strings are interpreted as positive numbers when assigned to INTEGERs or TimeTicks.

Object Syntax	Defval Syntax	Examples
INTEGER	<integer> or <hexString> or <bitString>	1 or -2 or '34'h or '10001'b
<enumerated>	<enumName>	up
TimeTicks	<integer> or <hexString> or <bitString>	1 or 'a45'h or '10111'b
OCTET STRING	<quotedString> or <hexString> or <bitString>	"example" or '123abf'h or '10100100'b
OBJECT IDENTIFIER	<objectName> or "{" <objectValue> "}"	sysDescr or { wellKnwTst loopback } or { 0 0 }
IpAddress	<hexString>	'c0210415'h

Figure 15 - Syntax and Examples of Defval values

4.5.13. Considerations for Instances

An SNMP variable is an object's identity and its instance value encoded as an OID. Objects that are not in a table (i.e., simple objects) are given an instance with a single component whose value is zero. For example, the SNMP variable for "sysDescr" (which is not in a table) is:

```
iso  org  dod  internet  mgmt  mib  system  sysDescr (instance)
 1    3    6      1       2     1     1        1        0

or  1.3.6.1.2.1.1.1.0  or  sysDescr.0
```

For objects within a table, the definition of the row object must indicate with the INDEX clause the columns and the order in which they must be used to specify instance values for items in the row. (Note: instances must be defined as columns within the table (or use the discouraged practice of a syntax type name).) Figure 16 gives the encoding rules for each base syntax type of an instance item. Examples of instance encodings are shown in Figure 17.

Syntax Type	**Instance Encoding**
Integer	A single component is used (note that only columns with nonnegative integer values are valid).
Fixed length string	"n" components are used, one for each byte in the string. The column must be defined as a fixed size octet string or a textual convention that resolves to a fixed size octet string.
Varying length string	"n+1" components are used. The first component has the string length and each byte of the string uses an OID component.
IP address	Four components are used. Each byte of the IP address in network order uses a component.
Object identifier	"n+1" components are used. The first is the number of components in the OID value. Each component in the value uses a component in the instance.
Network Address	Five components are used. The first component has the value "1" to indicate an IP address. The next four components store the four bytes of an IP address in network order.

NOTE: Some implementations that have a varying length string or an object identifier as the last instance component may not have a length component specified. The Concise MIB RFC needs to be updated to specify a syntax to indicate when this encoding is used. (Currently there is no syntax to specify this encoding with the INDEX clause.)

Figure 16 - Instance Encoding Rules

```
Examples:
                   object              object
                  -------- instance  --------------- instance
                  |      | |         |             | |
simple object   - sysDescr.0     or  1.3.6.1.2.1.1.1.0
(The instance is one component with value zero.)

                   object              object
                  ------- instance   ------------------ instance
                  |     | |          |                 | |
columnar object - ifSpeed.1      or  1.3.6.1.2.1.2.2.1.5.1
(The instance is one component with the integer value of ifIndex.)

                   object              instance
                  ------------ -----------------------
                  |          | |                     |
columnar object - tcpConnState.1.2.3.4.100.5.6.7.8.200
                               |     |  |  |     |   |
                               -------  |  -------   |
                                1st    2nd   3rd    4th
(The instance consists of four parts.  The first part is four
 components with the IP address value of tcpConnLocalAddress.
 The second part is one component with the integer value of
 tcpConnLocalPort.  The third part is four components with the IP
 address value of tcpConnRemAddress.  The fourth and last part
 is one component with the integer value of tcpConnRemPort.)

                  object          instance
                  ------------- -----------
                  |           | |         |
columnar object - atPhysAddress.3.1.4.5.6.7
                                | |       |
                                | ---------
                               1st    2nd
(The instance consists of two parts.  The first part is one
 component with the integer value of atIfIndex.  The second part
 is five components.  The first component is value 1 to indicate
 that an IP address follows.  The next four components are the
 IP address value for atNetAddress.)

                  object          instance
                  ------------ ---------------------
                  |          | |                   |
columnar object - partyTDomain.9.1.3.6.1.4.1.999.4.1
                               | |                 |
                               | -------------------
                           count      value
(The instance is an OID which is encoded as two parts.  The first
 part is one component which contains the count of the components
 in the OID.  The second part is the component values of the OID
 value for partyIdentity.)
```

Figure 17 - Examples of Instance Encodings

4.5.14. Guidelines for Objects

When defining objects, the following guidelines should be followed:

- Put objects into logical groups. Use hierarchical subgrouping for more detailed arrangement. Remember that a complete group may be designated as optional or mandatory. A placeholder object (which is not itself a managed object) may be defined as an OID for the group. The IETF MIB-II defines the following groups: system, interfaces, at, ip, icmp, tcp, udp, egp, transmission, and snmp.

- No SNMP-managed object defined in the "mib" subtree may have an OID component with a value of 0. Items that are not SNMP-managed objects that are identified by OIDs can use 0 as a component value. (Note: SNMP variables may have 0 as a component value in the instance part.)

- The OID for a table's row object should be one level below the table and have the last component with a value of 1. No other OIDs should be defined as siblings of the row object. The OIDs for the columns in the row should be one level below the row object.

- In SNMP, aggregate objects are defined as tables. One or more columns of the table are designated as the indices of the rows in the table. Tables cannot be defined within tables. Therefore, potentially nested tables should be elevated to the same level as the original table. Columns that are the indices from the original table should be added to the elevated table using a different name. The indices of the elevated table will be the "added and renamed" indices from the original table plus the natural indices. (The procedure to construct SNMP tables from more complex programming language data structures is not described in this article.)

- Tables that allow row creation and deletion should have a column named "xxxType" or "xxxStatus" which is an enumerated value. By convention, the first value should be called "other" (or "valid") and the second value should be called "invalid". (This is not always followed.) A row is removed by a single set operation that specifies the value of the "xxxType" (or "xxxStatus") variable as "invalid". A new row is added by using a single set operation. The variables in the set operation are the required columns in the new row using the new instance. The DESCRIPTION clause should describe the complete semantics of create and delete operations. More complex tables may have additional values for the "xxxType" (or "xxxStatus") column so that rows can be "constructed" using several set operations instead of just one. The RMON MIB[11] has "xxxStatus" objects in tables that may have four values, instead of two. One value called "underCreation" is used to indicate that the row exists, but the values of columns in the row may not be consistent and associated resources may not be allocated. This article does not cover these kinds of complex MIB tables.

- The DESCRIPTION clause should be included with each leaf object to describe its function and use.

- All names defined within a MIB module must be unique. Object names must start with a lowercase letter. The names of objects that are counters should end in the letter "s".

- Objects that are printable strings should be defined as DisplayString. Objects that contain pure binary information should be defined as octet strings.

- Objects that are media- or physical-level addresses should be defined as PhysAddress instead of as an octet string.

- Objects should be designed so that an SNMP operation can be performed twice in a row without disastrous results. SNMP runs on top of an unreliable transport service that may cause SNMP requests to be duplicated or responses to be dropped. Applications may retry when no response is received.

- The number of columns in a table should be kept small enough so that the entire row can be retrieved with one SNMP request operation.

4.5.15. General MIB Design Rules

The following guidelines, which are summarized from the the Concise MIB RFC[6], are general rules for deciding how many and which objects to manage:

- Too much information creates as much a problem as not enough information. Begin slowly and try to specify only the key objects to be managed.

- Start with the objects that are essential for fault or configuration management.

- Because of the current lack of a security system in SNMP with secure authentication and privacy, only weak control objects should be specified. (Security in SNMP has finally been defined and is waiting for IETF approval. This restriction may no longer be valid by mid 1992.)

- Objects must have demonstrated current use and should not be defined as placeholders for future implementation.

- Redundancy should be avoided by not defining variables that can be derived from others (for example, through arithmetic means.)

- Case diagrams[10] should be used to show the relationships between counters. (These proved invaluable in determining the counters for MIB-I[4].)

- Objects should be chosen that are general in nature and can be used for other products.

- Critical sections of code should not be heavily instrumented.

- After a device has network management added, it must still be able to function effectively in its primary role.

4.5.16. Case Diagrams

To aid in determining if sufficient, yet not redundant, counters have been specified to characterize a "flow", a visual diagram should be constructed. (These are called Case diagrams[10], after their inventor, Jeff Case.) In a protocol layer, the number of packets received from the layer below is equal to the number of packets received in error, plus the number of packets that are forwarded, plus the number of packets that were delivered to the layer above. The number of packets sent to the layer below is also equal to the number of packet requests from the layer above plus the number of forwarded packets. This relationship is shown in Figure 18.

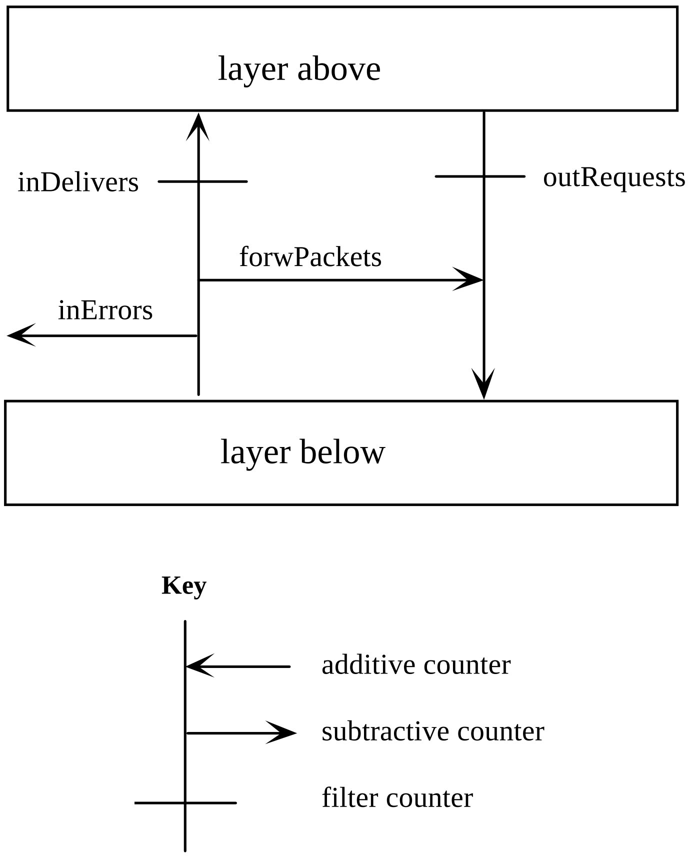

Figure 18 - Case Diagram

Case diagrams show the logical flow, not the actual implementation. In practice, the diagram can have additional complexity with several error counters, which may be incremented at any point in the implementation. Case diagrams augment, but do not replace, individual descriptions for each counter.

4.6. Traps

Traps are used to signal a manager that an extraordinary event has occurred at an agent. Unfortunately, the syntax for traps is one of the weaker points of the SNMP protocol. Instead of using OIDs to identify traps, a flat-numbering scheme was chosen for the six events associated with the MIB-I definitions, and an extension mechanism was specified. This mechanism is triggered when the generic-trap field has the value "enterpriseSpecific(6)". When this occurs, the values of the specific-trap and enterprise fields are used together to determine the event.

At present, the extension mechanism has seen limited implementation. No interoperability experiences have yet been published. Part of the reluctance to use this mechanism is due to the lack of agreement on the value of the "sysObjectID" MIB variable. The SNMP protocol calls for its value to be returned as the value of the enterprise field in traps. However, the current interpretation of the enterprise field has been changed in the Concise Trap RFC[7].

4.6.1. Trap-Type Macro

The Concise Trap RFC[7] gives guidance on the specification and interpretation of traps. This is done by providing an ASN.1 macro to define traps and clarifying the value that should be stored in the enterprise field. The trap macro is shown in Figure 19:

```
Syntax:
      <trapItem> = <trapName> "TRAP-TYPE"
            "ENTERPRISE" <oidName>
            [ "VARIABLES" "{" <varName> ["," <varName>]... "}" ]
            [ "DESCRIPTION" <description> ]
            [ "REFERENCE" <reference> ]
            "::=" <value>

Where:
      <trapName> is the name of the trap being defined;
      <oidName> is either "snmp" or the value to return in the
            "enterprise" field;
      <value> is the value of the trap returned in either the
            "generic-trap" or "specific-trap" fields; and
      Values for <varName>, <description>, and <reference> are
            defined later.
```

Figure 19 - Syntax for TRAP-TYPE Macro

Missing from the TRAP-TYPE macro, but present in the OBJECT-TYPE macro is the STATUS clause. Until this is added in a future version of the Concise Trap RFC, the status of the trap should be specified using an ASN.1 comment.

4.6.2. Values for ENTERPRISE

The required ENTERPRISE clause determines the value to be returned in the enterprise field of the returned trap. If the value specified in the macro is "snmp", then the value returned is the value of the sysObjectID object at the agent generating the trap (and the trap will be an SNMP-generic trap). If the value in the macro is not "snmp", then the value from the ENTERPRISE clause is the one that must be returned (and the trap must be an enterprise-specific trap). (See the definition of the "Values for Trap-Type" for additional information.)

4.6.3. Values for VARIABLES

The optional VARIABLES clause names the interesting SNMP-managed objects that must be returned in the trap. The DESCRIPTION clause must indicate the instance for each object to be returned. The agent implementer may choose to return additional variables. Care should be taken to choose variables so that the trap can be returned in no more than 484 octets. Network management systems must be able to interpret any returned variables, not just those specified in the trap definition.

4.6.4. Values for DESCRIPTION

The value for the optional DESCRIPTION clause is a textual definition of that trap that provides all semantic definitions necessary for implementation. It is meant as information for agent and manager writers and not as the "help text" for manager users. It is a quoted string just like the one for the DESCRIPTION clause in the OBJECT-TYPE macro.

4.6.5. Values for REFERENCE

The value for the REFERENCE clause is a "textual cross-reference" to another document that describes the same trap. This is used when converting MIBs from other formats, such as IEEE or OSI. It is a quoted string just like the one for the REFERENCE clause in the OBJECT-TYPE macro.

4.6.6. Values for TRAP-TYPE

The TRAP-TYPE macro value and the enterprise value together determine the values to be returned in the generic-trap and specific-trap fields in a trap. As previously specified, if the value for the ENTERPRISE clause is "snmp", then the trap being defined is a generic trap. In this case, the value for the trap-type macro is one of the values (i.e., 0 thru 5) for generic traps. This number is returned in the generic-trap field, and the specific-trap field is returned as zero. (Note: no new generic traps may be defined.) For specific traps, the trap-type macro specifies the value of the specific-trap field. The generic-trap field must be returned with value "enterpriseSpecific(6)" by the agent. See Figure 20 for an example of traps.

```
Examples:
      coldStart TRAP-TYPE
            ENTERPRISE snmp
            ::= 0


      fooTrap TRAP-TYPE
            ENTERPRISE foo
            ::= 45

Where
      coldStart is a generic trap;
      and fooTrap is an enterprise-specific trap.
```

Figure 20 - Examples for TRAP-TYPE Macro

4.6.7. Considerations for Traps

SNMP traps are not confirmed or uniquely identified (i.e., no request-id field). Therefore, it is not possible for an agent to determine if a manager has received a trap nor for a manager to determine if a trap is a duplicate. While agent logging, confirmation of trap receipt, and recognition of duplicate traps can be added, they are not currently part of the SNMP specification. The definition of an event log is also not in the current IETF MIBs. Thus, current managers and agents should not depend on confirmation of receipt, duplicate recognition, or agent logging to function properly. However, traps and associated objects should be designed to consume a small amount of agent resources, so that the manager will be able to recognize that an event has occurred via low-frequency polling. Some proprietary schemes that function between consenting agents and managers have been developed to overcome the deficiencies noted above. They are not interoperable between arbitrary pairs of managers and agents that do not know the conventions being followed.

5. Standard Practices Used in MIBs

The MIBs defined by the IETF working groups, starting from December 1990, have been written in the Concise MIB form. Very few "tricks" have been used. However, proprietary MIBs developed by vendors may still be in the original MIB format. This section describes some of the standard practices and "tricks" that may be found in proprietary and working group-developed MIBs. Over time, practices change. This section may contain dated information.

5.1. Modules

Modules are an ASN.1 mechanism used to group items. All names defined within a module must be unique. An IETF working group has not yet developed a MIB document with multiple MIB modules. It is quite valid to define multiple MIB modules within one MIB document. However, few of the popular network managers that allow MIB definitions as input allow more than one module to be specified. To create the input file for them, all the MIB modules of interest must be combined together with an editor into one giant MIB module. Because of this implementation restriction, MIB objects with the same name but different OID values may not be allowed. The names of objects may need to be changed to enable MIBs from different sources to be loaded. To reduce the probability of duplicate names, MIB objects should have a consistent prefix given to their names. By mid to late 1992, many managers should have the capability to support multiple MIB modules.

5.2. The IMPORTS Clause

The IMPORTS clause specifies the names of objects defined in other modules that are used in the current module. ASN.1 allows one module to reference an object defined in another module without the use of the IMPORTS clause by using the syntax <moduleName>"."<object>. For example, "RFC1155-SMI.experimental" could be used to reference the object named "experimental" from the RFC1155-SMI module. This is not a current practice in MIBs. Few of the popular network managers fully implement the IMPORTS clause. For these systems, all SMI names must be built into the MIB parsing component of the management system. By mid to late 1992, more managers should be upgraded to fully support the IMPORTS clause. However, it is unlikely that support of the dotted notation will be added.

5.3. The Object Identifier { 0 0 }

By convention, the object identifier { 0 0 } is being used as the default value for MIB objects that have the syntax type of OID, since the value for { 0 0 } is easily encoded using ASN.1 Basic Encoding Rules (BER). For example, in the Interface Extensions MIB[12], the "ifExtnsChipSet" is an object that identifies the hardware chip set being used on the interface. Its syntax type is defined as an object identifier. As currently specified, the object identifier of { 0 0 } is returned if the chip set is unknown. This causes some difficulties for the general-purpose network manager since the OID of { 0 0 } is used for other purposes. In the same MIB, the object "ifExtnsTestCode" is defined as containing specific information on a test result. As currently specified, the object identifier of { 0 0 } is returned if the test code is unknown. Thus, an appropriate description or name cannot be attached to the OID { 0 0 }. A clear direction has not been determined yet to either continue to special case the { 0 0 } OID or for other defaults to be used.

5.4. Textual Conventions

Some of the newer MIBs, such as the RMON MIB, are defining a section called textual conventions. This section is used as a timesaver throughout the MIB to define the semantics of syntax types in one place, instead of redundantly specifying them each time the syntax type is used. In a fully flexible approach, the SMI would be modified, when needed, to add additional syntax types. The SNMP SMI is essentially frozen. The textual conventions section is a conforming mechanism for extension of the SMI. Its disadvantage is that general-purpose management applications have no knowledge of the true semantics associated with the data types defined using this mechanism. For example, the RMON MIB[11] defines "EntryStatus" as an enumerated integer. Only applications especially designed for devices containing RMON MIBs will be able to interpret and display the proper value of fields defined with syntax type "EntryStatus". Until management station software is extended to correctly process textual conventions, MIBs that use them will generally need to have the base syntax type substituted into the syntax type definition to enable those MIBs to be read without errors by MIB parsers. There has been an explosion in the use of textual convention in MIBs developed by IETF working groups. This should force management station software to handle textual conventions in a generic and extensible method by mid to late 1992.

5.5. Object Creation and Deletion

SNMP does not have a *create* or *delete* operation. These functions may be performed via a *set* operation with proper MIB design. Creation and deletion are used on *instances* of conceptional rows in tables. The standard practice for new MIBs is to have a column in the table named "xxxType" or "xxxStatus" that has a syntax type of enumerated value. A row of the table is deleted by doing a *set* on the "xxxType" (or "xxxStatus") variable with a value of "invalid". Creation is carried out by doing a *set* that contains the required variables for the row to be created. Currently, there is no formal specification mechanism to describe the semantics of *create* and *delete*. The DESCRIPTION clause for the table should be used to indicate if *delete* and *create* are allowed and, if they are, to describe how they are implemented and the associated semantics. Older proprietary MIBs that allowed creation and/or deletion may have complex mechanisms that require special-purpose programs to implement the operations.

The new RMON MIB[11] has a need to allow multiple managers to create rows in a table at the same time. The simple mechanism described above was not sufficient. To be able to support creation by multiple managers, to support creation when all the columns could not be contained within one *set* request, and to allow modification of a row in a table without first deleting it, a new mechanism using four values was developed. Other IETF working groups are following the results of this new model of row creation and deletion.

5.6. Optional Objects

SNMP MIBs do not allow an object to be given the status of "optional". (See the section named "Value for Status" for rules.) A complete group such as the EGP group may be optional. This means that, if the device is performing EGP routing, the group must be present but the group must not be present if the device is not running EGP. From an agent standpoint, it may not be able to implement some objects. For example, if a network interface device driver has not been instrumented, it would be impossible for the agent to determine the number of packets received over that interface. The current practice is for agents to return the error of "noSuchName" for objects that can not be implemented. This unexpected result may cause difficulties in some applications that have been written to expect all mandatory objects to be implemented. To overcome this application problem, some agents have been designed to return benign values, such as zero, for objects that have not been implemented. This practice is discouraged; it does not change an agent from being non-conformant to a MIB group to being conformant.

5.7. OID Restrictions

There are no formal constraints specified for OIDs, which are used extensively through SNMP. ASN.1 allows OIDs of unlimited length with component values of unlimited size. Currently, no IETF MIB requires an OID component to be bigger than a 16-bit unsigned integer. However, one of the reference SNMP implementations uses an OID component that is a 32-bit unsigned integer, thus forcing management station support of components of at least this size. The current practice is to keep OID components limited to a 16 bit maximum. The total length of an OID has not yet been limited. However, for encoding efficiency, a limit of 127 bytes as the length is desirable. From a practical standpoint, the length of an SNMP request is limited to 484 bytes, since this is the minimum length that all SNMP agents and managers must implement. Use of long OIDs to identify objects reduces the number of SNMP variables in an SNMP message.

5.8. Proprietary Instance Rules

Until the Concise MIB RFC[6] was published, there were no written rules to provide guidance on instance specification. Several vendors have developed MIBs that have one or more objects that do not follow the instance rules. It is impossible to parse these MIBs and correctly determine the format of the instance specification for these objects. As a result, the objects can not be supported by the popular network management systems; they can only be supported by custom application programs. Technically these objects are in violation of the SMI as interpreted by the Concise MIB RFC.

6. Proprietary Extensions

Standard MIBs provide a starting place for a vendor to choose the managed objects to implement in a device or server program. SNMP allows a device (or program) to be both monitored and controlled. The standard MIBs define the generic objects for a device to be monitored, but provide few objects to control a device. (The RMON MIB[11] is one of the few examples of a MIB that provides enough control objects so that a RMON device can be completely managed via SNMP.) In addition to using standard MIBs "as is", vendors may create new objects in their proprietary MIB tree that describe the unique characteristics of the device and allow the vendor specific features to be monitored and controlled. Also, the vendors may wish to create objects to extend the monitoring and control of features covered by the standard MIBs. The new objects may not be added under the branches in the OID tree used for the standard MIB objects. They must be added under the branch assigned to the vendor (or enterprise) by the IANA.

Each vendor must determine the organization of its OID registration subtree. There is no single correct or proper way to construct the subtree; the breadth of the types of the devices (or programs) made by vendors typically influence the complexity of the subtree organization. One approach is to create four branches at the top of a vendor's subtree. The first branch is used to contain the registration OIDs returned as values for "sysObjectID". The second branch is used to contain the vendor's "experimental" MIB objects such as those for demos at trade shows or for experiments within the vendor's research and development groups. The third branch is used to create extensions to standard MIBs. The fourth branch contains proprietary MIB objects organized by major product lines or capabilities. Figure 21 shows one organization of a vendor's OID subtree.

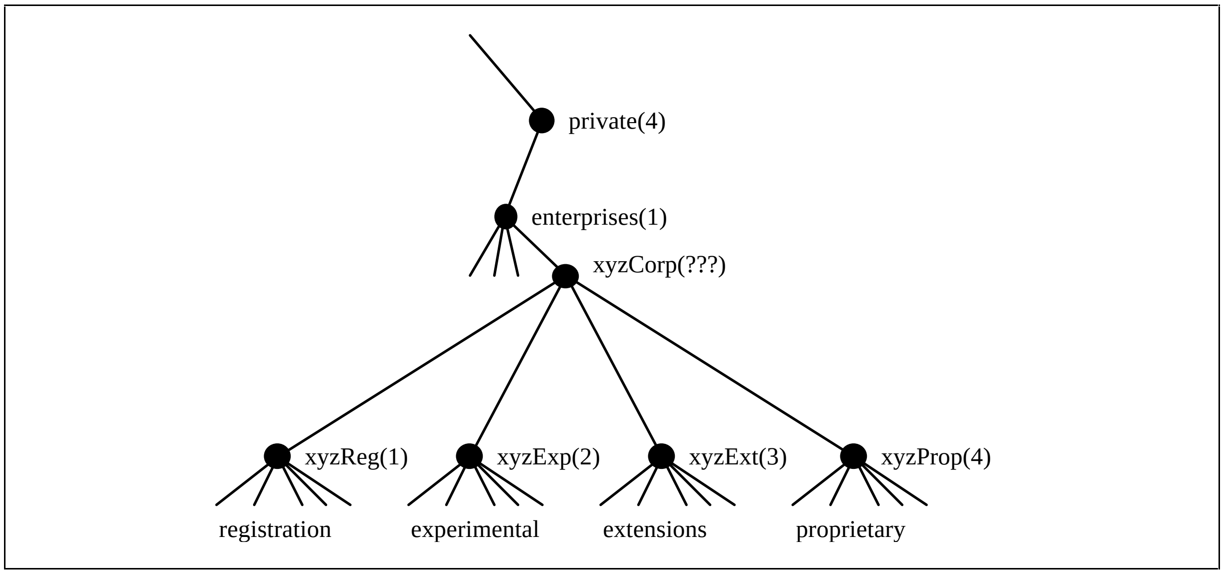

Figure 21 - Organization of a Vendor's (or Enterprise) Subtree

The most popular method to extend standard MIBs is to create shadow copies of the standard MIB tree structure. The new objects must be added in the enterprise's subtree and not in the standard MIB subtree. For example, suppose that extensions were to be added to the "system" and "IP" groups. To do this, branches would be added underneath the "xyzExt" branch and the new objects would be added under these branches. Figure 22 shows a fragment of the resulting MIB tree after adding objects to extend the system and IP groups. Visualizing the relationship between the standard objects and the extended objects may be difficult because they are in distant branches of the OID registration tree. Network management stations should allow logical grouping of related MIB objects; some do, others don't.

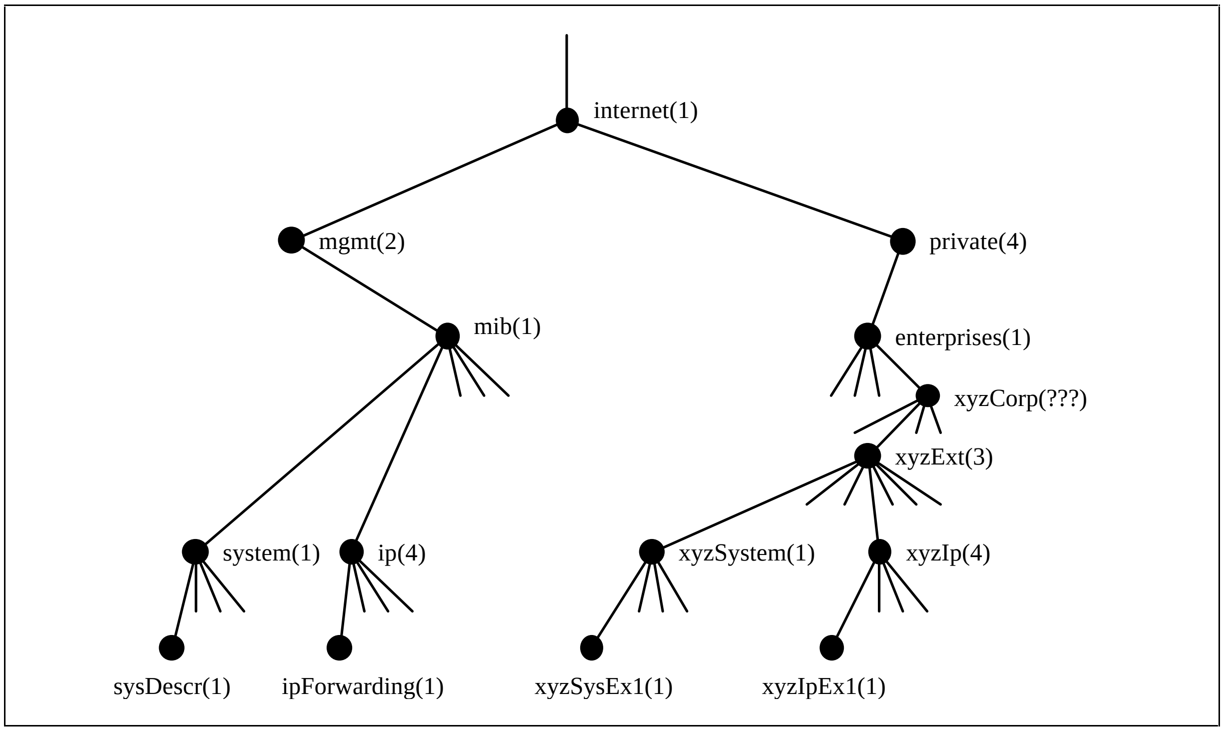

Figure 22 - Shadow MIB tree for proprietary extensions

There are two methods of extending tables. Both create shadow OIDs for the table and entry. The first method is to use the index variables from the original table. The second method is to create shadow copies of the index variables from the original table. The choice of method usually depends on the number of index variables. For example, the ifTable from MIB-II has ifIndex as its index variable. The ifTable, when extended, usually has a shadow copy of ifIndex created. On the other hand, the tcpConnTable from MIB-II which has four index objects does not typically have the index objects shadowed. Figure 23 shows a table using a new (or shadowed) index object and Figure 24 shows a table using existing index objects. Note that, in both cases, the index objects are listed in the INDEX clause for the row and are items of the sequence definition for the table.

```
xyzInterfaces OBJECT IDENTIFIER ::= { xyzExt 2 }

xyzIfTable OBJECT-TYPE
      SYNTAX SEQUENCE OF XyzIfEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { xyzInterfaces 1 }

xyzIfEntry OBJECT-TYPE
      SYNTAX XyzIfEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX  { xyzIfIndex }
      ::= { xyzIfTable 1 }

XyzIfEntry ::= SEQUENCE {
      xyzIfIndex  INTEGER,
      xyzIfProp1  Counter,
      ...(all other objects in table)
      }

xyzIfIndex OBJECT-TYPE
      SYNTAX INTEGER (1..65535)
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
            "Index for xyzIfTable"
      ::= { xyzIfEntry 1 }

xyzIfProp1 OBJECT-TYPE
      SYNTAX Counter
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
            "Proprietary extension to interfaces table"
      ::= { xyzIfEntry 2 }
```

Figure 23 - Example table using new index

```
xyzTcp OBJECT IDENTIFIER ::= { xyzExt 6 }

xyzTcpConnTable OBJECT-TYPE
      SYNTAX SEQUENCE OF XyzTcpConnEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { xyzTcp 1 }

xyzTcpConnEntry OBJECT-TYPE
      SYNTAX XyzTcpConnEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX  { tcpConnLocalAddress, tcpConnLocalPort,
               tcpConnRemAddress,   tcpConnRemPort }
      ::= { xyzTcpTable 1 }

XyzTcpConnEntry ::= SEQUENCE {
      tcpConnState            INTEGER,
      tcpConnLocalAddress     IpAddress,
      tcpConnLocalPort        INTEGER,
      tcpConnRemAddress       IpAddress,
      tcpConnRemPort          INTEGER,
      xyzTcpConnProp1         Counter,
      ...(all other objects in table)
      }

xyzTcpConnProp1 OBJECT-TYPE
      SYNTAX Counter
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
            "Proprietary extension to tcpConnTable"
      ::= { xyzTcpConnEntry 1 }
```

Figure 24 - Example table using existing index

7. Future Directions

Network management using SNMP is a process that is evolving continuously and rapidly. Even though the protocol and SMI documents have been essentially unchanged since 1988, the process of specifying MIB objects has affected their interpretation. The clumsiness of the original MIB specification format was replaced by the Concise MIB[6] and Concise Trap[7] formats. This article has pointed out some of the difficulties in these documents that need to be rectified. Hopefully, all the documents defining SNMP network management will be rewritten soon.

8. Topics Not Covered

Not all MIB topics have been covered in this article. An interesting topic for a future article is the detailed examination of row creation and deletion (some details have already been mentioned). The RMON MIB would be a place to use as an example to show the techniques and trade-offs in developing the MIB objects for row creation and deletion. Another interesting topic is how to design MIB objects used to perform actions. These types of objects exist in a few of the newer MIBs. A topic of much discussion is how to design large tables so they can be retrieved efficiently and quickly, but are still capable of fast lookups of rows. Finally, another topic of interest is how to specify MIB objects that correspond to programming objects of tables within tables (or arrays of structures within arrays of structures).

9. Sources for Documents and IANA Contact Information

```
RFCs and Internet-Drafts are available by anonymous FTP.  Login
with username "anonymous" and password "guest".  RFCs are located
in the rfc directory and Internet-Drafts are located in the
internet-drafts directory.  Use the "cd" command to position to
the directory of interest.  Use the "get" command to get one
document, or the "mget" command to get multiple documents.  The
file rfc-index.txt in the rfc directory and the file
1id-abstracts.txt in the internet-drafts directory contain a
description of the files in the respective directories.  The
following machines are available:

* East Coast (US): nnsc.nsf.net (128.89.1.178)

* West Coast (US): ftp.nisc.sri.com (192.33.33.22)

* Pacific Rim: munnari.oz.au (128.250.1.21) (compressed format)

* Europe: nic.nordu.net (192.36.148.17) (internet-drafts only)

Both RFCs and Internet-Drafts are available by EMail.  Send a
message to: mail-server@nisc.sri.com.  In the body enter one or
more "SEND" requests whose format is "SEND <directory>/<file>".

The most current information about obtaining RFCs via FTP or
EMAIL may be obtained by sending an EMAIL message to
"rfc-info@ISI.EDU" with the message body containing
"help: ways_to_get_rfcs".

The most current list of online libraries for RFCs is available in
the file "in-notes/rfc-retrieval.txt" on VENERA.ISI.EDU
(128.9.0.32).
```

Figure 25 - How to get an RFC or Internet-Draft

```
The Internet Assigned Numbers Authority can be reached at:

      Postal:     Joyce K. Reynolds
                  Internet Assigned Numbers Authority
                  USC/Information Sciences Institute
                  4676 Admiralty Way
                  Marina del Rey, CA 90292-6695
                  US

      Phone:      +1 213-822-1511

      Mail:       iana@isi.edu
```

Figure 26 - How to Contact the Internet Assigned Numbers Authority

```
Many standard and proprietary MIBs are available by anonymous FTP.
Login with username "anonymous" and password "guest".  The
following locations contain MIB files:

* venera.isi.edu (128.9.0.32) - mibs are located in
                                    directory mibs.
```

Figure 27 **-** Locations of MIBs on the Internet

David Perkins is an active member of the IETF network management community. He is also a member of the IETF Network Management Directoriate, and the author of the Standards Column in the SimpleTimes.

10. Acronyms Used In This Article

ASN.1: Abstract Syntax Notation One
BER: Basic Encoding Rules
BGP: Border Gateway Protocol
CMIP: Common Management Information Protocol
CMOT: CMIP over TCP/IP
EGP: Exterior Gateway Protocol
IANA: Internet Assigned Numbers Authority
IEEE: Institute of Electronic and Electrical Engineers
IETF: Internet Engineering Task Force
MIB: Management Information Base
OID: Object Identifier
OSI: Open Systems Interconnection
PDU: Protocol Data Unit
RFC: Request For Comment
RMON: Remote Network Monitoring
SMI: Structure of Management Information
SNMP: Simple Network Management Protocol

11. References

1. *The Simple Book: An Introduction to Management of TCP/IP-based Internets*, Marshall T. Rose, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

2. *Simple Network Management Protocol*, Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, C.Marshall T. Rose and Keith McCloghrie, RFC 1157 (previously RFC 1098 and RFC 1067), May 1990.

3. *Structure and Identification of Management Information for TCP/IP-based Internets*, Marshall T. Rose and Keith McCloghrie, RFC 1155 (previously RFC 1065), May 1990.

4. *Management Information Base for Network Management of TCP/IP based Internets*, Keith McCloghrie and Marshall T. Rose, RFC 1156 (previously RFC 1066), May 1990. Note: this document is obsoleted by RFC 1213.

5. *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, Marshall T. Rose (editor), RFC 1213 (previously RFC 1158), March 1991.

6. *Towards Concise MIB Definitions*, Keith McCloghrie and Marshall T. Rose (editors), RFC 1212, March 1991.

7. *A Convention for Defining Traps for use with the SNMP*, Marshall T. Rose (editor), RFC 1215, March 1991.

8. *IAB recommendations for the development of Internet management standards*, Vint Cerf, RFC 1052, April 1988.

9. *Report of the second Ad Hoc Network Management Review Group*, Vint Cerf, RFC 1109, August 1989.

10. "Case Diagrams: A first Step to Diagrammed Management Information Bases," *Computer Communication Review*, Jeffrey D. Case and Craig Partridge, 19(1):13-16, January, 1989.

11. *Remote network monitoring Management Information Base*, Steve Waldbusser, RFC 1271, November 1991.

12. *Extensions to the generic-interface MIB*, Keith McCloghrie (editor), RFC 1229 (updated by RFC 1239), May 1991.

13. *Reassignment of experimental MIBs to standard MIBs*, Joyce Reynolds, RFC 1239, June 1991.

14. *IAB official protocol standards*, Jon Postel (editor), RFC 1250, August 1991.