Network Working Group
Request for Comments: 1155
Obsoletes: RFC 1065

M. Rose
Performance Systems International
K. McCloghrie
Hughes LAN Systems

May 1990

Structure and Identification of Management Information for TCP/IP-based Internets

Table of Contents

1. Status of this Memo

This RFC is a re-release of RFC 1065, with a changed "Status of this Memo", plus a few minor typographical corrections. The technical content of the document is unchanged from RFC 1065.

This memo provides the common definitions for the structure and identification of management information for TCP/IP-based internets. In particular, together with its companion memos which describe the management information base along with the network management protocol, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular, the Internet.

This memo specifies a Standard Protocol for the Internet community. Its status is "Recommended". TCP/IP implementations in the Internet which are network manageable are expected to adopt and implement this specification.

The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least one of the two recommended management protocols SNMP (RFC-1157) or CMOT (RFC-1095). It should be noted that, at this time, SNMP is a full Internet standard and CMOT is a draft standard. See also the Host and Gateway Requirements RFCs for more specific information on the applicability of this standard.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

2. Introduction

This memo describes the common structures and identification scheme for the definition of management information used in managing TCP/IP-based internets. Included are descriptions of an object information model for network management along with a set of generic types used to describe management information. Formal descriptions of the structure are given using Abstract Syntax Notation One (ASN.1) [1].

This memo is largely concerned with organizational concerns and administrative policy: it neither specifies the objects which are managed, nor the protocols used to manage those objects. These concerns are addressed by two companion memos: one describing the Management Information Base (MIB) [2], and the other describing the Simple Network Management Protocol (SNMP) [3].

This memo is based in part on the work of the Internet Engineering

Task Force, particularly the working note titled "Structure and Identification of Management Information for the Internet" [4]. This memo uses a skeletal structure derived from that note, but differs in one very significant way: that note focuses entirely on the use of OSI-style network management. As such, it is not suitable for use with SNMP.

This memo attempts to achieve two goals: simplicity and extensibility. Both are motivated by a common concern: although the management of TCP/IP-based internets has been a topic of study for some time, the authors do not feel that the depth and breadth of such understanding is complete. More bluntly, we feel that previous experiences, while giving the community insight, are hardly conclusive. By fostering a simple SMI, the minimal number of constraints are imposed on future potential approaches; further, by fostering an extensible SMI, the maximal number of potential approaches are available for experimentation.

It is believed that this memo and its two companions comply with the guidelines set forth in RFC 1052, "IAB Recommendations for the Development of Internet Network Management Standards" [5] and RFC 1109, "Report of the Second Ad Hoc Network Management Review Group" [6]. In particular, we feel that this memo, along with the memo describing the management information base, provide a solid basis for network management of the Internet.

3. Structure and Identification of Management Information

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1) [1].

Each type of object (termed an object type) has a name, a syntax, and an encoding. The name is represented uniquely as an OBJECT IDENTIFIER. An OBJECT IDENTIFIER is an administratively assigned name. The administrative policies used for assigning names are discussed later in this memo.

The syntax for an object type defines the abstract data structure corresponding to that object type. For example, the structure of a given object type might be an INTEGER or OCTET STRING. Although in general, we should permit any ASN.1 construct to be available for use in defining the syntax of an object type, this memo purposely restricts the ASN.1 constructs which may be used. These restrictions are made solely for the sake of simplicity.

The encoding of an object type is simply how instances of that object type are represented using the object's type syntax. Implicitly tied to the notion of an object's syntax and encoding is how the object is represented when being transmitted on the network. This memo specifies the use of the basic encoding rules of ASN.1 [7].

It is beyond the scope of this memo to define either the MIB used for network management or the network management protocol. As mentioned earlier, these tasks are left to companion memos. This memo attempts to minimize the restrictions placed upon its companions so as to maximize generality. However, in some cases, restrictions have been made (e.g., the syntax which may be used when defining object types in the MIB) in order to encourage a particular style of management. Future editions of this memo may remove these restrictions.

3.1. Names

Names are used to identify managed objects. This memo specifies names which are hierarchical in nature. The OBJECT IDENTIFIER concept is used to model this notion. An OBJECT IDENTIFIER can be used for purposes other than naming managed object types; for example, each international standard has an OBJECT IDENTIFIER assigned to it for the purposes of identification. In short, OBJECT IDENTIFIERs are a means for identifying some object, regardless of the semantics associated with the object (e.g., a network object, a standards document, etc.)

An OBJECT IDENTIFIER is a sequence of integers which traverse a global tree. The tree consists of a root connected to a number of labeled nodes via edges. Each node may, in turn, have children of its own which are labeled. In this case, we may term the node a subtree. This process may continue to an arbitrary level of depth. Central to the notion of the OBJECT IDENTIFIER is the understanding that administrative control of the meanings assigned to the nodes may be delegated as one traverses the tree. A label is a pairing of a brief textual description and an integer.

The root node itself is unlabeled, but has at least three children directly under it: one node is administered by the International Organization for Standardization, with label iso(1); another is administrated by the International Telegraph and Telephone Consultative Committee, with label ccitt(0); and the third is jointly administered by the ISO and the CCITT, joint-iso-ccitt(2).

Under the iso(1) node, the ISO has designated one subtree for use by other (inter)national organizations, org(3). Of the children nodes present, two have been assigned to the U.S. National Institutes of Standards and Technology. One of these subtrees has been transferred by the NIST to the U.S. Department of Defense, dod(6).

As of this writing, the DoD has not indicated how it will manage its subtree of OBJECT IDENTIFIERs. This memo assumes that DoD will allocate a node to the Internet community, to be administered by the Internet Activities Board (IAB) as follows:

```
internet    OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

That is, the Internet subtree of OBJECT IDENTIFIERs starts with the prefix:

```
1.3.6.1.
```

This memo, as a standard approved by the IAB, now specifies the policy under which this subtree of OBJECT IDENTIFIERs is administered. Initially, four nodes are present:

```
directory     OBJECT IDENTIFIER ::= { internet 1 }
mgmt          OBJECT IDENTIFIER ::= { internet 2 }
experimental  OBJECT IDENTIFIER ::= { internet 3 }
private       OBJECT IDENTIFIER ::= { internet 4 }
```

3.1.1. Directory

The directory(1) subtree is reserved for use with a future memo that discusses how the OSI Directory may be used in the Internet.

3.1.2. Mgmt

The mgmt(2) subtree is used to identify objects which are defined in IAB-approved documents. Administration of the mgmt(2) subtree is delegated by the IAB to the Internet Assigned Numbers Authority for the Internet. As RFCs which define new versions of the Internet-standard Management Information Base are approved, they are assigned an OBJECT IDENTIFIER by the Internet Assigned Numbers Authority for identifying the objects defined by that memo.

For example, the RFC which defines the initial Internet standard MIB would be assigned management document number 1. This RFC would use the OBJECT IDENTIFIER

```
{ mgmt 1 }
```

or

```
1.3.6.1.2.1
```

in defining the Internet-standard MIB.

The generation of new versions of the Internet-standard MIB is a rigorous process. Section 5 of this memo describes the rules used when a new version is defined.

3.1.3. Experimental

The experimental(3) subtree is used to identify objects used in Internet experiments. Administration of the experimental(3) subtree is delegated by the IAB to the Internet Assigned Numbers Authority of the Internet.

For example, an experimenter might received number 17, and would have available the OBJECT IDENTIFIER

```
{ experimental 17 }
```

or

```
1.3.6.1.3.17
```

for use.

As a part of the assignment process, the Internet Assigned Numbers Authority may make requirements as to how that subtree is used.

3.1.4. Private

The private(4) subtree is used to identify objects defined unilaterally. Administration of the private(4) subtree is delegated by the IAB to the Internet Assigned Numbers Authority for the Internet. Initially, this subtree has at least one child:

```
enterprises   OBJECT IDENTIFIER ::= { private 1 }
```

The enterprises(1) subtree is used, among other things, to permit parties providing networking subsystems to register models of their products.

Upon receiving a subtree, the enterprise may, for example, define new MIB objects in this subtree. In addition, it is strongly recommended that the enterprise will also register its networking subsystems under this subtree, in order to provide an unambiguous identification mechanism for use in management protocols. For example, if the "Flintstones, Inc." enterprise produced networking subsystems, then they could request a node under the enterprises subtree from the Internet Assigned Numbers Authority. Such a node might be numbered:

```
1.3.6.1.4.1.42
```

The "Flintstones, Inc." enterprise might then register their "Fred Router" under the name of:

```
1.3.6.1.4.1.42.1.1
```

3.2. Syntax

Syntax is used to define the structure corresponding to object types. ASN.1 constructs are used to define this structure, although the full generality of ASN.1 is not permitted.

The ASN.1 type ObjectSyntax defines the different syntaxes which may be used in defining an object type.

3.2.1. Primitive Types

Only the ASN.1 primitive types INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL are permitted. These are sometimes referred to as non-aggregate types.

3.2.1.1. Guidelines for Enumerated INTEGERs

If an enumerated INTEGER is listed as an object type, then a named-number having the value 0 shall not be present in the list of enumerations. Use of this value is prohibited.

3.2.2. Constructor Types

The ASN.1 constructor type SEQUENCE is permitted, providing that it is used to generate either lists or tables.

For lists, the syntax takes the form:

```
SEQUENCE { <type1>, ..., <typeN> }
```

where each <type> resolves to one of the ASN.1 primitive types listed above. Further, these ASN.1 types are always present (the DEFAULT and OPTIONAL clauses do not appear in the SEQUENCE definition).

For tables, the syntax takes the form:

```
SEQUENCE OF <entry>
```

where <entry> resolves to a list constructor.

Lists and tables are sometimes referred to as aggregate types.

3.2.3. Defined Types

In addition, new application-wide types may be defined, so long as they resolve into an IMPLICITly defined ASN.1 primitive type, list, table, or some other application-wide type. Initially, few application-wide types are defined. Future memos will no doubt define others once a consensus is reached.

3.2.3.1. NetworkAddress

This CHOICE represents an address from one of possibly several protocol families. Currently, only one protocol family, the Internet family, is present in this CHOICE.

3.2.3.2. IpAddress

This application-wide type represents a 32-bit internet address. It is represented as an OCTET STRING of length 4, in network byte-order.

When this ASN.1 type is encoded using the ASN.1 basic encoding rules, only the primitive encoding form shall be used.

3.2.3.3. Counter

This application-wide type represents a non-negative integer which monotonically increases until it reaches a maximum value, when it wraps around and starts increasing again from zero. This memo specifies a maximum value of 2^32-1 (4294967295 decimal) for counters.

3.2.3.4. Gauge

This application-wide type represents a non-negative integer, which may increase or decrease, but which latches at a maximum value. This memo specifies a maximum value of 2^32-1 (4294967295 decimal) for gauges.

3.2.3.5. TimeTicks

This application-wide type represents a non-negative integer which counts the time in hundredths of a second since some epoch. When object types are defined in the MIB which use this ASN.1 type, the description of the object type identifies the reference epoch.

3.2.3.6. Opaque

This application-wide type supports the capability to pass arbitrary ASN.1 syntax. A value is encoded using the ASN.1 basic rules into a string of octets. This, in turn, is encoded as an OCTET STRING, in effect "double-wrapping" the original ASN.1 value.

Note that a conforming implementation need only be able to accept and recognize opaquely-encoded data. It need not be able to unwrap the data and then interpret its contents.

Further note that by use of the ASN.1 EXTERNAL type, encodings other than ASN.1 may be used in opaquely-encoded data.

3.3. Encodings

Once an instance of an object type has been identified, its value may be transmitted by applying the basic encoding rules of ASN.1 to the syntax for the object type.

4. Managed Objects

Although it is not the purpose of this memo to define objects in the MIB, this memo specifies a format to be used by other memos which define these objects.

An object type definition consists of five fields:

OBJECT:

A textual name, termed the OBJECT DESCRIPTOR, for the object type, along with its corresponding OBJECT IDENTIFIER.

Syntax:
The abstract syntax for the object type. This must resolve to an instance of the ASN.1 type ObjectSyntax (defined below).

Definition:
A textual description of the semantics of the object type. Implementations should ensure that their instance of the object fulfills this definition since this MIB is intended for use in multi-vendor environments. As such it is vital that objects have consistent meaning across all machines.

Access:
One of read-only, read-write, write-only, or not-accessible.

Status:
One of mandatory, optional, or obsolete.

Future memos may also specify other fields for the objects which they define.

4.1. Guidelines for Object Names

No object type in the Internet-Standard MIB shall use a sub-identifier of 0 in its name. This value is reserved for use with future extensions.

Each OBJECT DESCRIPTOR corresponding to an object type in the internet-standard MIB shall be a unique, but mnemonic, printable string. This promotes a common language for humans to use when discussing the MIB and also facilitates simple table mappings for user interfaces.

4.2. Object Types and Instances

An object type is a definition of a kind of managed object; it is declarative in nature. In contrast, an object instance is an instantiation of an object type which has been bound to a value. For example, the notion of an entry in a routing table might be defined in the MIB. Such a notion corresponds to an object type; individual entries in a particular routing table which exist at some time are object instances of that object type.

A collection of object types is defined in the MIB. Each such subject type is uniquely named by its OBJECT IDENTIFIER and also has a textual name, which is its OBJECT DESCRIPTOR. The means whereby object instances are referenced is not defined in the MIB. Reference to object instances is achieved by a protocol-specific mechanism: it is the responsibility of each management protocol adhering to the SMI to define this mechanism.

An object type may be defined in the MIB such that an instance of that object type represents an aggregation of information also represented by instances of some number of "subordinate" object types. For example, suppose the following object types are defined in the MIB:

```
OBJECT:
-------
   atIndex { atEntry 1 }

Syntax:
   INTEGER

Definition:
   The interface number for the physical address.

Access:
   read-write.

Status:
   mandatory.


OBJECT:
-------
   atPhysAddress { atEntry 2 }

Syntax:
   OCTET STRING

Definition:
   The media-dependent physical address.
Access:
   read-write.

Status:
   mandatory.


OBJECT:
-------
   atNetAddress { atEntry 3 }

Syntax:
   NetworkAddress

Definition:
   The network address corresponding to the media-dependent physical
   address.

Access:
   read-write.

Status:
   mandatory.

Then, a fourth object type might also be defined in the MIB:


OBJECT:
-------
   atEntry { atTable 1 }

Syntax:

   AtEntry ::= SEQUENCE {
         atIndex
         INTEGER,
         atPhysAddress
         OCTET STRING,
         atNetAddress
         NetworkAddress
         }

Definition:
   An entry in the address translation table.

Access:
   read-write.
Status:
   mandatory.
```

Each instance of this object type comprises information represented by instances of the former three object types. An object type defined in this way is called a list.

Similarly, tables can be formed by aggregations of a list type. For example, a fifth object type might also be defined in the MIB:

```
   OBJECT:
   ------
      atTable { at 1 }

   Syntax:
      SEQUENCE OF AtEntry

   Definition:
      The address translation table.

   Access:
      read-write.

   Status:
      mandatory.
```

such that each instance of the atTable object comprises information represented by the set of atEntry object types that collectively constitute a given atTable object instance, that is, a given address translation table.

Consider how one might refer to a simple object within a table. Continuing with the previous example, one might name the object type

```
      { atPhysAddress }
```

and specify, using a protocol-specific mechanism, the object instance

```
      { atNetAddress } = { internet "10.0.0.52" }
```

This pairing of object type and object instance would refer to all instances of atPhysAddress which are part of any entry in some address translation table for which the associated atNetAddress value is { internet "10.0.0.52" }.

To continue with this example, consider how one might refer to an aggregate object (list) within a table. Naming the object type

```
      { atEntry }
```

and specifying, using a protocol-specific mechanism, the object instance

```
      { atNetAddress } = { internet "10.0.0.52" }
```

refers to all instances of entries in the table for which the associated atNetAddress value is { internet "10.0.0.52" }.

Each management protocol must provide a mechanism for accessing simple (non-aggregate) object types. Each management protocol specifies whether or not it supports access to aggregate object types. Further, the protocol must specify which instances are "returned" when an object type/instance pairing refers to more than one instance of a type.

To afford support for a variety of management protocols, all information by which instances of a given object type may be usefully distinguished, one from another, is represented by instances of object types defined in the MIB.

4.3. Macros for Managed Objects

In order to facilitate the use of tools for processing the definition of the MIB, the OBJECT-TYPE macro may be used. This macro permits the key aspects of an object type to be represented in a formal way.

```
      OBJECT-TYPE MACRO ::=
      BEGIN
          TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                            "ACCESS" Access
                            "STATUS" Status
          VALUE NOTATION ::= value (VALUE ObjectName)

          Access ::= "read-only"
                          | "read-write"
                          | "write-only"
                          | "not-accessible"
          Status ::= "mandatory"
                          | "optional"
                          | "obsolete"
          END
```

Given the object types defined earlier, we might imagine the following definitions being present in the MIB:

```
                  atIndex OBJECT-TYPE
```

```
                          SYNTAX  INTEGER
                          ACCESS  read-write
                          STATUS  mandatory
                          ::= { atEntry 1 }

                  atPhysAddress OBJECT-TYPE
                          SYNTAX  OCTET STRING
                          ACCESS  read-write
                          STATUS  mandatory
                          ::= { atEntry 2 }

                  atNetAddress OBJECT-TYPE
                          SYNTAX  NetworkAddress
                          ACCESS  read-write
                          STATUS  mandatory
                          ::= { atEntry 3 }

                  atEntry OBJECT-TYPE
                          SYNTAX  AtEntry
                          ACCESS  read-write
                          STATUS  mandatory
                          ::= { atTable 1 }

                  atTable OBJECT-TYPE
                          SYNTAX  SEQUENCE OF AtEntry
                          ACCESS  read-write
                          STATUS  mandatory
                          ::= { at 1 }

                  AtEntry ::= SEQUENCE {
                      atIndex
                          INTEGER,
                      atPhysAddress
                          OCTET STRING,
                      atNetAddress
                          NetworkAddress
                  }
```

The first five definitions describe object types, relating, for example, the OBJECT DESCRIPTOR atIndex to the OBJECT IDENTIFIER { atEntry 1 }. In addition, the syntax of this object is defined (INTEGER) along with the access permitted (read-write) and status (mandatory). The sixth definition describes an ASN.1 type called AtEntry.

5. Extensions to the MIB

Every Internet-standard MIB document obsoletes all previous such documents. The portion of a name, termed the tail, following the OBJECT IDENTIFIER

```
   { mgmt version-number }
```

used to name objects shall remain unchanged between versions. New versions may:

(1) declare old object types obsolete (if necessary), but not delete their names;

(2) augment the definition of an object type corresponding to a list by appending non-aggregate object types to the object types in the list; or,

(3) define entirely new object types.

New versions may not:

(1) change the semantics of any previously defined object without changing the name of that object.

These rules are important because they admit easier support for multiple versions of the Internet-standard MIB. In particular, the semantics associated with the tail of a name remain constant throughout different versions of the MIB. Because multiple versions of the MIB may thus coincide in "tail-space," implementations supporting multiple versions of the MIB can be vastly simplified.

However, as a consequence, a management agent might return an instance corresponding to a superset of the expected object type. Following the principle of robustness, in this exceptional case, a manager should ignore any additional information beyond the definition of the expected object type. However, the robustness principle requires that one exercise care with respect to control actions: if an instance does not have the same syntax as its expected object type, then those control actions must fail. In both the monitoring and control cases, the name of an object returned by an operation must be identical to the name requested by an operation.

6. Definitions

```
          RFC1155-SMI DEFINITIONS ::= BEGIN

          EXPORTS -- EVERYTHING
                  internet, directory, mgmt,
                  experimental, private, enterprises,
                  OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
                  ApplicationSyntax, NetworkAddress, IpAddress,
                  Counter, Gauge, TimeTicks, Opaque;

           -- the path to the root

           internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }

           directory     OBJECT IDENTIFIER ::= { internet 1 }

           mgmt          OBJECT IDENTIFIER ::= { internet 2 }

           experimental  OBJECT IDENTIFIER ::= { internet 3 }

           private       OBJECT IDENTIFIER ::= { internet 4 }
           enterprises   OBJECT IDENTIFIER ::= { private 1 }


           -- definition of object types

           OBJECT-TYPE MACRO ::=
           BEGIN
               TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                                 "ACCESS" Access
                                 "STATUS" Status
               VALUE NOTATION ::= value (VALUE ObjectName)

               Access ::= "read-only"
                               | "read-write"
                               | "write-only"
                               | "not-accessible"
               Status ::= "mandatory"
                               | "optional"
                               | "obsolete"
           END

              -- names of objects in the MIB

              ObjectName ::=
                  OBJECT IDENTIFIER
              -- syntax of objects in the MIB

              ObjectSyntax ::=
                  CHOICE {
                      simple
                          SimpleSyntax,

              -- note that simple SEQUENCEs are not directly
              -- mentioned here to keep things simple (i.e.,
              -- prevent mis-use).  However, application-wide
              -- types which are IMPLICITly encoded simple
              -- SEQUENCEs may appear in the following CHOICE

                      application-wide
                          ApplicationSyntax
                  }

                 SimpleSyntax ::=
                     CHOICE {
                         number
                             INTEGER,

                         string
                             OCTET STRING,

                         object
                             OBJECT IDENTIFIER,

                         empty
                             NULL
                     }

                 ApplicationSyntax ::=
                     CHOICE {
                         address
                             NetworkAddress,

                         counter
                             Counter,

                         gauge
                             Gauge,

                         ticks
                             TimeTicks,

                         arbitrary
                             Opaque
                 -- other application-wide types, as they are
                 -- defined, will be added here
                     }


                 -- application-wide types

                 NetworkAddress ::=
                     CHOICE {
                         internet
                             IpAddress
                     }

                 IpAddress ::=
                     [APPLICATION 0]          -- in network-byte order
                         IMPLICIT OCTET STRING (SIZE (4))

                 Counter ::=
                     [APPLICATION 1]
                         IMPLICIT INTEGER (0..4294967295)

                 Gauge ::=
                     [APPLICATION 2]
                         IMPLICIT INTEGER (0..4294967295)

                 TimeTicks ::=
                     [APPLICATION 3]
                         IMPLICIT INTEGER (0..4294967295)

                 Opaque ::=
                     [APPLICATION 4]          -- arbitrary ASN.1 value,
                         IMPLICIT OCTET STRING   --   "double-wrapped"

                 END
```

7. Acknowledgements

This memo was influenced by three sets of contributors to earlier drafts:

First, Lee Labarre of the MITRE Corporation, who as author of the NETMAN SMI [4], presented the basic roadmap for the SMI.

Second, several individuals who provided valuable comments on this memo prior to its initial distribution:

James R. Davin, Proteon
Mark S. Fedor, NYSERNet
Craig Partridge, BBN Laboratories
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Wengyik Yeong, NYSERNet

Third, the IETF MIB working group:

Karl Auerbach, Epilogue Technology
K. Ramesh Babu, Excelan
Lawrence Besaw, Hewlett-Packard
Jeffrey D. Case, University of Tennessee at Knoxville
James R. Davin, Proteon
Mark S. Fedor, NYSERNet
Robb Foster, BBN
Phill Gross, The MITRE Corporation
Bent Torp Jensen, Convergent Technology
Lee Labarre, The MITRE Corporation
Dan Lynch, Advanced Computing Environments
Keith McCloghrie, The Wollongong Group
Dave Mackie, 3Com/Bridge
Craig Partridge, BBN (chair)
Jim Robertson, 3Com/Bridge
Marshall T. Rose, The Wollongong Group
Greg Satz, cisco
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Lou Steinberg, IBM
Dean Throop, Data General
Unni Warrier, Unisys

8. References

[1] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[2] McCloghrie K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets", RFC 1156, Performance Systems International and Hughes LAN Systems, May 1990.

[3] Case, J., M. Fedor, M. Schoffstall, and J. Davin, The Simple Network Management Protocol", RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance Systems International, and the MIT Laboratory for Computer Science, May 1990.

[4] LaBarre, L., "Structure and Identification of Management Information for the Internet", Internet Engineering Task Force working note, Network Information Center, SRI International, Menlo Park, California, April 1988.

[5] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[6] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.

[7] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International Organization for Standardization, International Standard 8825, December 1987.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Marshall T. Rose
PSI, Inc.
PSI California Office
P.O. Box 391776
Mountain View, CA 94039

Phone: (415) 961-3380

EMail: mrose@PSI.COM

Keith McCloghrie
The Wollongong Group
1129 San Antonio Road
Palo Alto, CA 04303

Phone: (415) 962-7160

EMail: sytek!kzm@HPLABS.HP.COM

==

```
Network Working Group                                            M. Rose
Request for Comments: 1212          Performance Systems International
                                                          K. McCloghrie
                                                     Hughes LAN Systems
                                                                Editors
                                                             March 1991
```

Concise MIB Definitions

Status of this Memo

This memo defines a format for producing MIB modules. This RFC specifies an IAB standards track document for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Table of Contents

1. Abstract

This memo describes a straight-forward approach toward producing concise, yet descriptive, MIB modules. It is intended that all future MIB modules be written in this format.

2. Historical Perspective

As reported in RFC 1052, IAB Recommendations for the Development of Internet Network Management Standards [1], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP), defined in RFC 1067, was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: RFC 1065, which defined the Structure of Management Information (SMI), and RFC 1066, which defined the Management Information Base (MIB). Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC 1109, Report of the Second Ad Hoc Network Management Review Group [2], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, based on the SNMP, to respond to new operational needs in the Internet community by producing MIB-II.

In May of 1990, the core documents were elevated to "Standard Protocols" with "Recommended" status. As such, the Internet-standard network management framework consists of: Structure and Identification of Management Information for TCP/IP-based internets, RFC 1155 [3], which describes how managed objects contained in the

MIB are defined; Management Information Base for Network Management of TCP/IP-based internets, which describes the managed objects contained in the MIB, RFC 1156 [4]; and, the Simple Network Management Protocol, RFC 1157 [5], which defines the protocol used to manage these objects. Consistent with the IAB directive to produce simple, workable systems in the short-term, the list of managed objects defined in the Internet-standard MIB was derived by taking only those elements which are considered essential. However, the SMI defined three extensibility mechanisms: one, the addition of new standard objects through the definitions of new versions of the MIB; two, the addition of widely-available but non-standard objects through the experimental subtree; and three, the addition of private objects through the enterprises subtree. Such additional objects can not only be used for vendor-specific elements, but also for experimentation as required to further the knowledge of which other objects are essential.

As more objects are defined using the second method, experience has shown that the resulting MIB descriptions contain redundant information. In order to provide for MIB descriptions which are more concise, and yet as informative, an enhancement is suggested. This enhancement allows the author of a MIB to remove the redundant information, while retaining the important descriptive text.

Before presenting the approach, a brief presentation of columnar object handling by the SNMP is necessary. This explains and further motivates the value of the enhancement.

3. Columnar Objects

The SNMP supports operations on MIB objects whose syntax is ObjectSyntax as defined in the SMI. Informally stated, SNMP operations apply exclusively to scalar objects. However, it is convenient for developers of management applications to impose imaginary, tabular structures on the ordered collection of objects that constitute the MIB. Each such conceptual table contains zero or more rows, and each row may contain one or more scalar objects, termed columnar objects. Historically, this conceptualization has been formalized by using the OBJECT-TYPE macro to define both an object which corresponds to a table and an object which corresponds to a row in that table. (The ACCESS clause for such objects is "not-accessible", of course.) However, it must be emphasized that, at the protocol level, relationships among columnar objects in the same row is a matter of convention, not of protocol.

Note that there are good reasons why the tabular structure is not a matter of protocol. Consider the operation of the SNMP Get-Next-PDU acting on the last columnar object of an instance of a conceptual row; it returns the next column of the first conceptual row or the first object instance occurring after the table. In contrast, if the rows were a matter of protocol, then it would instead return an error. By not returning an error, a single PDU exchange informs the manager that not only has the end of the conceptual row/table been reached, but also provides information on the next object instance, thereby increasing the information density of the PDU exchange.

3.1. Row Deletion

Nonetheless, it is highly useful to provide a means whereby a conceptual row may be removed from a table. In MIB-II, this was achieved by defining, for each conceptual row, an integer-valued columnar object. If a management station sets the value of this object to some value, usually termed "invalid", then the effect is one of invalidating the corresponding row in the table. However, it is an implementation-specific matter as to whether an agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the columnar object indicating the in-use status.

3.2. Row Addition

It is also highly useful to have a clear understanding of how a conceptual row may be added to a table. In the SNMP, at the protocol level, a management station issues an SNMP set operation containing an arbitrary set of variable bindings. In the case that an agent detects that one or more of those variable bindings refers to an object instance not currently available in that agent, it may, according to the rules of the SNMP, behave according to any of the following paradigms:

(1) It may reject the SNMP set operation as referring to non-existent object instances by returning a response with the error-status field set to "noSuchName" and the error-index field set to refer to the first vacuous reference.

(2) It may accept the SNMP set operation as requesting the creation of new object instances corresponding to each of the object instances named in the variable bindings. The value of each (potentially) newly created object instance is specified by the "value" component of the relevant variable binding. In this case, if the request specifies a value for a newly (or previously) created object that it deems inappropriate by reason of value or

syntax, then it rejects the SNMP set operation by responding with the error-status field set to badValue and the error-index field set to refer to the first offending variable binding.

(3) It may accept the SNMP set operation and create new object instances as described in (2) above and, in addition, at its discretion, create supplemental object instances to complete a row in a conceptual table of which the new object instances specified in the request may be a part.

It should be emphasized that all three of the above behaviors are fully conformant to the SNMP specification and are fully acceptable, subject to any restrictions which may be imposed by access control and/or the definitions of the MIB objects themselves.

4. Defining Objects

The Internet-standard SMI employs a two-level approach towards object definition. A MIB definition consists of two parts: a textual part, in which objects are placed into groups, and a MIB module, in which objects are described solely in terms of the ASN.1 macro OBJECT-TYPE, which is defined by the SMI.

An example of the former definition might be:

```
OBJECT:
-------
     sysLocation { system 6 }

Syntax:
     DisplayString (SIZE (0..255))

Definition:
     The physical location of this node (e.g., "telephone
     closet, 3rd floor").

Access:
     read-only.

Status:
     mandatory.
```

An example of the latter definition might be:

```
sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
```

```
                   ACCESS  read-only
                   STATUS  mandatory
                   ::= { system 6 }
```

In the interests of brevity and to reduce the chance of editing errors, it would seem useful to combine the two definitions. This can be accomplished by defining an extension to the OBJECT-TYPE macro:

```
          IMPORTS
              ObjectName
                  FROM RFC1155-SMI
              DisplayString
                  FROM RFC1158-MIB;

          OBJECT-TYPE MACRO ::=
          BEGIN
              TYPE NOTATION ::=
                                          -- must conform to
                                          -- RFC1155's ObjectSyntax
                                "SYNTAX" type(ObjectSyntax)
                                "ACCESS" Access
                                "STATUS" Status
                                DescrPart
                                ReferPart
                                IndexPart
                                DefValPart
              VALUE NOTATION ::= value (VALUE ObjectName)

              Access ::= "read-only"
                              | "read-write"
                              | "write-only"
                              | "not-accessible"
              Status ::= "mandatory"
                              | "optional"
                              | "obsolete"
                              | "deprecated"

              DescrPart ::=
                         "DESCRIPTION" value (description DisplayString)
                              | empty

              ReferPart ::=
                         "REFERENCE" value (reference DisplayString)
                              | empty

              IndexPart ::=
                         "INDEX" "{" IndexTypes "}"
```

```
                                | empty
                IndexTypes ::=
                           IndexType | IndexTypes "," IndexType
                IndexType ::=
                                    -- if indexobject, use the SYNTAX
                                    -- value of the correspondent
                                    -- OBJECT-TYPE invocation
                           value (indexobject ObjectName)
                                    -- otherwise use named SMI type
                                    -- must conform to IndexSyntax below
                                | type (indextype)

                DefValPart ::=
                           "DEFVAL" "{" value (defvalue ObjectSyntax) "}"
                                | empty

            END

            IndexSyntax ::=
                CHOICE {
                    number
                        INTEGER (0..MAX),
                    string
                        OCTET STRING,
                    object
                        OBJECT IDENTIFIER,
                    address
                        NetworkAddress,
                    ipAddress
                        IpAddress
                }
```

4.1. Mapping of the OBJECT-TYPE macro

It should be noted that the expansion of the OBJECT-TYPE macro is something which conceptually happens during implementation and not during run-time.

4.1.1. Mapping of the SYNTAX clause

The SYNTAX clause, which must be present, defines the abstract data structure corresponding to that object type. The ASN.1 language [6] is used for this purpose. However, the SMI purposely restricts the ASN.1 constructs which may be used. These restrictions are made expressly for simplicity.

4.1.2. Mapping of the ACCESS clause

The ACCESS clause, which must be present, defines the minimum level of support required for that object type. As a local matter, implementations may support other access types (e.g., an implementation may elect to permitting writing a variable marked as read-only). Further, protocol-specific "views" (e.g., those indirectly implied by an SNMP community) may make further restrictions on access to a variable.

4.1.3. Mapping of the STATUS clause

The STATUS clause, which must be present, defines the implementation support required for that object type.

4.1.4. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which need not be present, contains a textual definition of that object type which provides all semantic definitions necessary for implementation, and should embody any information which would otherwise be communicated in any ASN.1 commentary annotations associated with the object. Note that, in order to conform to the ASN.1 syntax, the entire value of this clause must be enclosed in double quotation marks, although the value may be multi-line.

Further, note that if the MIB module does not contain a textual description of the object type elsewhere then the DESCRIPTION clause must be present.

4.1.5. Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a textual cross-reference to an object defined in some other MIB module. This is useful when de-osifying a MIB produced by some other organization.

4.1.6. Mapping of the INDEX clause

The INDEX clause, which may be present only if that object type corresponds to a conceptual row, defines instance identification information for that object type. (Historically, each MIB definition contained a section entitled "Identification of OBJECT instances for use with the SNMP". By using the INDEX clause, this section need no longer occur as this clause concisely captures the precise semantics needed for instance identification.)

If the INDEX clause is not present, and the object type corresponds to a non-columnar object, then instances of the object are identified by appending a sub-identifier of zero to the name of that object. Further, note that if the MIB module does not contain a textual description of how instance identification information is derived for columnar objects, then the INDEX clause must be present.

To define the instance identification information, determine which object value(s) will unambiguously distinguish a conceptual row. The syntax of those objects indicate how to form the instance-identifier:

(1) integer-valued: a single sub-identifier taking the integer value (this works only for non-negative integers);

(2) string-valued, fixed-length strings: ‘n’ sub-identifiers, where ‘n’ is the length of the string (each octet of the string is encoded in a separate sub-identifier);

(3) string-valued, variable-length strings: ‘n+1’ sub-identifiers, where ‘n’ is the length of the string (the first sub-identifier is ‘n’ itself, following this, each octet of the string is encoded in a separate sub-identifier);

(4) object identifier-valued: ‘n+1’ sub-identifiers, where ‘n’ is the number of sub-identifiers in the value (the first sub-identifier is ‘n’ itself, following this, each sub-identifier in the value is copied);

(5) NetworkAddress-valued: ‘n+1’ sub-identifiers, where ‘n’ depends on the kind of address being encoded (the first sub-identifier indicates the kind of address, value 1 indicates an IpAddress); or,

(6) IpAddress-valued: 4 sub-identifiers, in the familiar a.b.c.d notation.

Note that if an "indextype" value is present (e.g., INTEGER rather than ifIndex), then a DESCRIPTION clause must be present; the text contained therein indicates the semantics of the "indextype" value.

By way of example, in the context of MIB-II [7], the following INDEX clauses might be present:

```
              objects under         INDEX clause
            -----------------       ------------
            ifEntry                 { ifIndex }
            atEntry                 { atNetIfIndex,
                                      atNetAddress }
            ipAddrEntry             { ipAdEntAddr }
            ipRouteEntry            { ipRouteDest }
            ipNetToMediaEntry       { ipNetToMediaIfIndex,
                                      ipNetToMediaNetAddress }
            tcpConnEntry            { tcpConnLocalAddress,
                                      tcpConnLocalPort,
                                      tcpConnRemoteAddress,
                                      tcpConnRemotePort }
            udpEntry                { udpLocalAddress,
                                      udpLocalPort }
            egpNeighEntry           { egpNeighAddr }
```

4.1.7. Mapping of the DEFVAL clause

The DEFVAL clause, which need not be present, defines an acceptable default value which may be used when an object instance is created at the discretion of the agent acting in conformance with the third paradigm described in Section 4.2 above.

During conceptual row creation, if an instance of a columnar object is not present as one of the operands in the correspondent SNMP set operation, then the value of the DEFVAL clause, if present, indicates an acceptable default value that the agent might use.

The value of the DEFVAL clause must, of course, correspond to the SYNTAX clause for the object. Note that if an operand to the SNMP set operation is an instance of a read-only object, then the error noSuchName will be returned. As such, the DEFVAL clause can be used to provide an acceptable default value that the agent might use.

It is possible that no acceptable default value may exist for any of the columnar objects in a conceptual row for which the creation of new object instances is allowed. In this case, the objects specified in the INDEX clause must have a corresponding ACCESS clause value of read-write.

By way of example, consider the following possible DEFVAL clauses:

ObjectSyntax	DEFVAL clause
INTEGER	1 -- same for Counter, Gauge, TimeTicks
OCTET STRING	’ffffffffffff’h
DisplayString	"any NVT ASCII string"
OBJECT IDENTIFIER	sysDescr
OBJECT IDENTIFIER	{ system 2 }
NULL	NULL
NetworkAddress	{ internet ’c0210415’h }
IpAddress	’c0210415’h -- 192.33.4.21

4.1.8. Mapping of the OBJECT-TYPE value

The value of an invocation of the OBJECT-TYPE macro is the name of the object, which is an object identifier.

4.2. Usage Example

Consider how the ipNetToMediaTable from MIB-II might be fully described:

```
        -- the IP Address Translation tables

        -- The Address Translation tables contain IpAddress to
        -- "physical" address equivalences.  Some interfaces do not
        -- use translation tables for determining address equivalences
        -- (e.g., DDN-X.25 has an algorithmic method); if all
        -- interfaces are of this type, then the Address Translation
        -- table is empty, i.e., has zero entries.

        ipNetToMediaTable OBJECT-TYPE
            SYNTAX  SEQUENCE OF IpNetToMediaEntry
            ACCESS  not-accessible
            STATUS  mandatory
            DESCRIPTION
                    "The IP Address Translation table used for mapping
                    from IP addresses to physical addresses."
            ::= { ip 22 }

        ipNetToMediaEntry OBJECT-TYPE
            SYNTAX  IpNetToMediaEntry
            ACCESS  not-accessible
            STATUS  mandatory
            DESCRIPTION
                    "Each entry contains one IpAddress to ’physical’
```

```
                address equivalence."
        INDEX   { ipNetToMediaIfIndex,
                  ipNetToMediaNetAddress }
        ::= { ipNetToMediaTable 1 }

    IpNetToMediaEntry ::=
        SEQUENCE {
            ipNetToMediaIfIndex
                INTEGER,
            ipNetToMediaPhysAddress
                OCTET STRING,
            ipNetToMediaNetAddress
                IpAddress,
            ipNetoToMediaType
                INTEGER
        }

    ipNetToMediaIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "The interface on which this entry's equivalence
                is effective.  The interface identified by a
                particular value of this index is the same
                interface as identified by the same value of
                ifIndex."
        ::= { ipNetToMediaEntry 1 }

    ipNetToMediaPhysAddress OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "The media-dependent 'physical' address."
        ::= { ipNetToMediaEntry 2 }

    ipNetToMediaNetAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "The IpAddress corresponding to the media-
                dependent 'physical' address."
        ::= { ipNetToMediaEntry 3 }

    ipNetToMediaType OBJECT-TYPE
        SYNTAX  INTEGER {
```

```
                          other(1),   -- none of the following
                          invalid(2), -- an invalidated mapping
                          dynamic(3),
                          static(4)
                      }
              ACCESS  read-write
              STATUS  mandatory
              DESCRIPTION
                      "The type of mapping.

                      Setting this object to the value invalid(2) has
                      the effect of invalidating the corresponding entry
                      in the ipNetToMediaTable.  That is, it effectively
                      disassociates the interface identified with said
                      entry from the mapping identified with said entry.
                      It is an implementation-specific matter as to
                      whether the agent removes an invalidated entry
                      from the table.  Accordingly, management stations
                      must be prepared to receive tabular information
                      from agents that corresponds to entries not
                      currently in use.  Proper interpretation of such
                      entries requires examination of the relevant
                      ipNetToMediaType object."
                  ::= { ipNetToMediaEntry 4 }
```

5. Appendix: DE-osifying MIBs

There has been an increasing amount of work recently on taking MIBs defined by other organizations (e.g., the IEEE) and de-osifying them for use with the Internet-standard network management framework. The steps to achieve this are straight-forward, though tedious. Of course, it is helpful to already be experienced in writing MIB modules for use with the Internet-standard network management framework.

The first step is to construct a skeletal MIB module, e.g.,

```
               RFC1213-MIB DEFINITIONS ::= BEGIN

               IMPORTS
                       experimental, OBJECT-TYPE, Counter
                           FROM RFC1155-SMI;

                       -- contact IANA for actual number
               root    OBJECT IDENTIFIER ::= { experimental xx }

               END
```

The next step is to categorize the objects into groups. For experimental MIBs, optional objects are permitted. However, when a MIB module is placed in the Internet-standard space, these optional objects are either removed, or placed in a optional group, which, if implemented, all objects in the group must be implemented. For the first pass, it is wisest to simply ignore any optional objects in the original MIB: experience shows it is better to define a core MIB module first, containing only essential objects; later, if experience demands, other objects can be added.

It must be emphasized that groups are "units of conformance" within a MIB: everything in a group is "mandatory" and implementations do either whole groups or none.

5.1. Managed Object Mapping

Next for each managed object class, determine whether there can exist multiple instances of that managed object class. If not, then for each of its attributes, use the OBJECT-TYPE macro to make an equivalent definition.

Otherwise, if multiple instances of the managed object class can exist, then define a conceptual table having conceptual rows each containing a columnar object for each of the managed object class's attributes. If the managed object class is contained within the containment tree of another managed object class, then the assignment of an object type is normally required for each of the "distinguished attributes" of the containing managed object class. If they do not already exist within the MIB module, then they can be added via the definition of additional columnar objects in the conceptual row corresponding to the contained managed object class.

In defining a conceptual row, it is useful to consider the optimization of network management operations which will act upon its columnar objects. In particular, it is wisest to avoid defining more columnar objects within a conceptual row, than can fit in a single PDU. As a rule of thumb, a conceptual row should contain no more than approximately 20 objects. Similarly, or as a way to abide by the "20 object guideline", columnar objects should be grouped into tables according to the expected grouping of network management operations upon them. As such, the content of conceptual rows should reflect typical access scenarios, e.g., they should be organized along functional lines such as one row for statistics and another row for parameters, or along usage lines such as commonly-needed objects versus rarely-needed objects.

On the other hand, the definition of conceptual rows where the number of columnar objects used as indexes outnumbers the number used to hold information, should also be avoided. In particular, the splitting of a managed object class's attributes into many conceptual tables should not be used as a way to obtain the same degree of flexibility/complexity as is often found in MIB's with a myriad of optionals.

5.1.1. Mapping to the SYNTAX clause

When mapping to the SYNTAX clause of the OBJECT-type macro:

(1) An object with BOOLEAN syntax becomes an INTEGER taking either of values true(1) or false(2).

(2) An object with ENUMERATED syntax becomes an INTEGER, taking any of the values given.

(3) An object with BIT STRING syntax containing no more than 32 bits becomes an INTEGER defined as a sum; otherwise if more than 32 bits are present, the object becomes an OCTET STRING, with the bits numbered from left-to-right, in which the least significant bits of the last octet may be "reserved for future use".

(4) An object with a character string syntax becomes either an OCTET STRING or a DisplayString, depending on the repertoire of the character string.

(5) An non-tabular object with a complex syntax, such as REAL or EXTERNAL, must be decomposed, usually into an OCTET STRING (if sensible). As a rule, any object with a complicated syntax should be avoided.

(6) Tabular objects must be decomposed into rows of columnar objects.

5.1.2. Mapping to the ACCESS clause

This is straight-forward.

5.1.3. Mapping to the STATUS clause

This is usually straight-forward; however, some osified-MIBs use the term "recommended". In this case, a choice must be made between "mandatory" and "optional".

5.1.4. Mapping to the DESCRIPTION clause

This is straight-forward: simply copy the text, making sure that any embedded double quotation marks are sanitized (i.e., replaced with single-quotes or removed).

5.1.5. Mapping to the REFERENCE clause

This is straight-forward: simply include a textual reference to the object being mapped, the document which defines the object, and perhaps a page number in the document.

5.1.6. Mapping to the INDEX clause

Decide how instance-identifiers for columnar objects are to be formed and define this clause accordingly.

5.1.7. Mapping to the DEFVAL clause

Decide if a meaningful default value can be assigned to the object being mapped, and if so, define the DEFVAL clause accordingly.

5.2. Action Mapping

Actions are modeled as read-write objects, in which writing a particular value results in the action taking place.

5.2.1. Mapping to the SYNTAX clause

Usually an INTEGER syntax is used with a distinguished value provided for each action that the object provides access to. In addition, there is usually one other distinguished value, which is the one returned when the object is read.

5.2.2. Mapping to the ACCESS clause

Always use read-write.

5.2.3. Mapping to the STATUS clause

This is straight-forward.

5.2.4. Mapping to the DESCRIPTION clause

This is straight-forward: simply copy the text, making sure that any embedded double quotation marks are sanitized (i.e., replaced with single-quotes or removed).

5.2.5. Mapping to the REFERENCE clause

This is straight-forward: simply include a textual reference to the action being mapped, the document which defines the action, and perhaps a page number in the document.

6. Acknowledgements

This document was produced by the SNMP Working Group:

Anne Ambler, Spider
Karl Auerbach, Sun
Fred Baker, ACC
Ken Brinkerhoff
Ron Broersma, NOSC
Jack Brown, US Army
Theodore Brunner, Bellcore
Jeffrey Buffum, HP
John Burress, Wellfleet
Jeffrey D. Case, University of Tennessee at Knoxville
Chris Chiptasso, Spartacus
Paul Ciarfella, DEC
Bob Collet
John Cook, Chipcom
Tracy Cox, Bellcore
James R. Davin, MIT-LCS
Eric Decker, cisco
Kurt Dobbins, Cabletron
Nadya El-Afandi, Network Systems
Gary Ellis, HP
Fred Engle
Mike Erlinger
Mark S. Fedor, PSI
Richard Fox, Synoptics
Karen Frisa, CMU
Chris Gunner, DEC
Fred Harris, University of Tennessee at Knoxville
Ken Hibbard, Xylogics
Ole Jacobsen, Interop
Ken Jones
Satish Joshi, Synoptics
Frank Kastenholz, Racal-Interlan
Shimshon Kaufman, Spartacus
Ken Key, University of Tennessee at Knoxville
Jim Kinder, Fibercom
Alex Koifman, BBN
Christopher Kolb, PSI
Cheryl Krupczak, NCR
Paul Langille, DEC
Peter Lin, Vitalink
John Lunny, TWG

Carl Malamud
Randy Mayhew, University of Tennessee at Knoxville
Keith McCloghrie, Hughes LAN Systems
Donna McMaster, David Systems
Lynn Monsanto, Sun
Dave Perkins, 3COM
Jim Reinstedler, Ungerman Bass
Anil Rijsinghani, DEC
Kathy Rinehart, Arnold AFB
Kary Robertson
Marshall T. Rose, PSI (chair)
L. Michael Sabo, NCSC
Jon Saperia, DEC
Greg Satz, cisco
Martin Schoffstall, PSI
John Seligson
Steve Sherry, Xyplex
Fei Shu, NEC
Sam Sjogren, TGV
Mark Sleeper, Sparta
Lance Sprung
Mike St.Johns
Bob Stewart, Xyplex
Emil Sturniold
Kaj Tesink, Bellcore
Dean Throop, Data General
Bill Townsend, Xylogics
Maurice Turcotte, Racal-Milgo
Kannan Varadhou
Sudhanshu Verma, HP
Bill Versteeg, Network Research Corporation
Warren Vik, Interactive Systems
David Waitzman, BBN
Steve Waldbusser, CMU
Dan Wintringhan
David Wood
Wengyik Yeong, PSI
Jeff Young, Cray Research

7. References

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, NRI, April 1988.

[2] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, NRI, August 1989.

[3] Rose M., and K. McCloghrie, "Structure and Identification of

Management Information for TCP/IP-based internets", RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[4] McCloghrie K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1156, Hughes LAN Systems, Performance Systems International, May 1990.

[5] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[6] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization International Standard 8824, December 1987.

[7] Rose M., Editor, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, Performance Systems International, March 1991.

8. Security Considerations

Security issues are not discussed in this memo.

9. Authors’ Addresses

Marshall T. Rose
Performance Systems International
5201 Great America Parkway
Suite 3106
Santa Clara, CA 95054

Phone: +1 408 562 6222
EMail: mrose@psi.com
X.500: rose, psi, us

Keith McCloghrie
Hughes LAN Systems
1225 Charleston Road
Mountain View, CA 94043
1225 Charleston Road
Mountain View, CA 94043

Phone: (415) 966-7934
EMail: kzm@hls.com