Network Working Group   J. Case
Request for Comments: 1442   SNMP Research, Inc.
                                                   K. McCloghrie
                                              Hughes LAN Systems
                                                        M. Rose
                                        Dover Beach Consulting, Inc.
                                                   S. Waldbusser
                                        Carnegie Mellon University
                                                     April 1993

                  Structure of Management Information
                        for version 2 of the
            Simple Network Management Protocol (SNMPv2)

Status of this Memo

This RFC specifes an IAB standards track protocol 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. Introduction

A network management system contains: several (potentially
many) nodes, each with a processing entity, termed an agent,
which has access to management instrumentation; at least one
management station; and, a management protocol, used to convey
management information between the agents and management
stations. Operations of the protocol are carried out under an
administrative framework which defines both authentication and
authorization policies.

Network management stations execute management applications
which monitor and control network elements. Network elements
are devices such as hosts, routers, terminal servers, etc.,
which are monitored and controlled through access to their
management information.

Management information is viewed as a collection of managed
objects, residing in a virtual information store, termed the
Management Information Base (MIB). Collections of related
objects are defined in MIB modules. These modules are written
using a subset of OSI's Abstract Syntax Notation One (ASN.1)
[1]. It is the purpose of this document, the Structure of
Management Information (SMI), to define that subset.

The SMI is divided into three parts: module definitions,
object definitions, and, trap definitions.

(1) Module definitions are used when describing information
     modules. An ASN.1 macro, MODULE-IDENTITY, is used to
     concisely convey the semantics of an information module.

(2) Object definitions are used when describing managed
     objects. An ASN.1 macro, OBJECT-TYPE, is used to
     concisely convey the syntax and semantics of a managed
     object.

(3) Notification definitions are used when describing
     unsolicited transmissions of management information. An
     ASN.1 macro, NOTIFICATION-TYPE, is used to concisely
     convey the syntax and semantics of a notification.

1.1. A Note on Terminology

For the purpose of exposition, the original Internet-standard
Network Management Framework, as described in RFCs 1155, 1157,
and 1212, is termed the SNMP version 1 framework (SNMPv1).
The current framework is termed the SNMP version 2 framework
(SNMPv2).

2. Definitions

SNMPv2-SMI DEFINITIONS ::= BEGIN

-- the path to the root

internet   OBJECT IDENTIFIER ::= { iso 3 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 }

security   OBJECT IDENTIFIER ::= { internet 5 }

snmpV2   OBJECT IDENTIFIER ::= { internet 6 }

-- transport domains
snmpDomains   OBJECT IDENTIFIER ::= { snmpV2 1 }

-- transport proxies
snmpProxys   OBJECT IDENTIFIER ::= { snmpV2 2 }

-- module identities
snmpModules   OBJECT IDENTIFIER ::= { snmpV2 3 }

```
-- definitions for information modules

MODULE-IDENTITY MACRO ::=
BEGIN
    TYPE NOTATION ::=
                    "LAST-UPDATED" value(Update UTCTime)
                    "ORGANIZATION" Text
                    "CONTACT-INFO" Text
                    "DESCRIPTION" Text
                    RevisionPart

    VALUE NOTATION ::=
                    value(VALUE OBJECT IDENTIFIER)

    RevisionPart ::=
                    Revisions
                  | empty
    Revisions ::=
                    Revision
                  | Revisions Revision
    Revision ::=
                    "REVISION" value(Update UTCTime)
                    "DESCRIPTION" Text

    -- uses the NVT ASCII character set
    Text ::= """" string """"
END
```

```
OBJECT-IDENTITY MACRO ::=
BEGIN
    TYPE NOTATION ::=
                "STATUS" Status
                "DESCRIPTION" Text
                ReferPart

    VALUE NOTATION ::=
                value(VALUE OBJECT IDENTIFIER)

    Status ::=
                "current"
              | "obsolete"

    ReferPart ::=
                "REFERENCE" Text
              | empty

    Text ::= """" string """"
END
```

```
-- names of objects

ObjectName ::=
    OBJECT IDENTIFIER


-- syntax of objects

ObjectSyntax ::=
    CHOICE {
        simple
            SimpleSyntax,

          -- note that SEQUENCEs for conceptual tables and
          -- rows are not mentioned here...

        application-wide
            ApplicationSyntax
    }


-- built-in ASN.1 types

SimpleSyntax ::=
    CHOICE {
        -- INTEGERs with a more restrictive range
        -- may also be used
        integer-value
            INTEGER (-2147483648..2147483647),

        string-value
            OCTET STRING,

        objectID-value
            OBJECT IDENTIFIER,

        -- only the enumerated form is allowed
        bit-value
            BIT STRING
    }
```

```
    -- indistinguishable from INTEGER, but never needs more than
    -- 32-bits for a two's complement representation
    Integer32 ::=
        [UNIVERSAL 2]
            IMPLICIT INTEGER (-2147483648..2147483647)


    -- application-wide types

    ApplicationSyntax ::=
        CHOICE {
            ipAddress-value
                IpAddress,

            counter-value
                Counter32,

            gauge-value
                Gauge32,

            timeticks-value
                TimeTicks,

            arbitrary-value
                Opaque,

            nsapAddress-value
                NsapAddress,

            big-counter-value
                Counter64,

            unsigned-integer-value
                UInteger32
        }

    -- in network-byte order
    -- (this is a tagged type for historical reasons)
    IpAddress ::=
        [APPLICATION 0]
            IMPLICIT OCTET STRING (SIZE (4))
```

```
-- this wraps
Counter32 ::=
    [APPLICATION 1]
        IMPLICIT INTEGER (0..4294967295)

-- this doesn't wrap
Gauge32 ::=
    [APPLICATION 2]
        IMPLICIT INTEGER (0..4294967295)

-- hundredths of seconds since an epoch
TimeTicks ::=
    [APPLICATION 3]
        IMPLICIT INTEGER (0..4294967295)

-- for backward-compatibility only
Opaque ::=
    [APPLICATION 4]
        IMPLICIT OCTET STRING

-- for OSI NSAP addresses
-- (this is a tagged type for historical reasons)
NsapAddress ::=
    [APPLICATION 5]
        IMPLICIT OCTET STRING (SIZE (1 | 4..21))

-- for counters that wrap in less than one hour with only 32 bits
Counter64 ::=
    [APPLICATION 6]
        IMPLICIT INTEGER (0..18446744073709551615)

-- an unsigned 32-bit quantity
UInteger32 ::=
    [APPLICATION 7]
        IMPLICIT INTEGER (0..4294967295)
```

```
-- definition for objects

OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::=
                "SYNTAX" type(Syntax)
                UnitsPart
                "MAX-ACCESS" Access
                "STATUS" Status
                "DESCRIPTION" Text
                ReferPart
                IndexPart
                DefValPart

    VALUE NOTATION ::=
                value(VALUE ObjectName)

    UnitsPart ::=
                "UNITS" Text
              | empty

    Access ::=
                "not-accessible"
              | "read-only"
              | "read-write"
              | "read-create"

    Status ::=
                "current"
              | "deprecated"
              | "obsolete"

    ReferPart ::=
                "REFERENCE" Text
              | empty

    IndexPart ::=
                "INDEX"     "{" IndexTypes "}"
              | "AUGMENTS" "{" Entry       "}"
              | empty
    IndexTypes ::=
                IndexType
              | IndexTypes "," IndexType
```

```
    IndexType ::=
                  "IMPLIED" Index
               | Index
    Index ::=
                   -- use the SYNTAX value of the
                   -- correspondent OBJECT-TYPE invocation
               value(Indexobject ObjectName)
    Entry ::=
                   -- use the INDEX value of the
                   -- correspondent OBJECT-TYPE invocation
               value(Entryobject ObjectName)

    DefValPart ::=
                  "DEFVAL" "{" value(Defval Syntax) "}"
               | empty

    -- uses the NVT ASCII character set
    Text ::= """" string """"
END
```

```
-- definitions for notifications

NOTIFICATION-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::=
                ObjectsPart
                "STATUS" Status
                "DESCRIPTION" Text
                ReferPart

    VALUE NOTATION ::=
                value(VALUE OBJECT IDENTIFIER)

    ObjectsPart ::=
                "OBJECTS" "{" Objects "}"
              | empty
    Objects ::=
                Object
              | Objects "," Object
    Object ::=
                value(Name ObjectName)

    Status ::=
                "current"
              | "deprecated"
              | "obsolete"

    ReferPart ::=
                "REFERENCE" Text
              | empty

    -- uses the NVT ASCII character set
    Text ::= """" string """"
END


END
```

3. Information Modules

An "information module" is an ASN.1 module defining
information relating to network management.

The SMI describes how to use a subset of ASN.1 to define an
information module. Further, additional restrictions are
placed on "standard" information modules. It is strongly
recommended that "enterprise-specific" information modules
also adhere to these restrictions.

Typically, there are three kinds of information modules:

(1)   MIB modules, which contain definitions of inter-related
      managed objects, make use of the OBJECT-TYPE and
      NOTIFICATION-TYPE macros;

(2)   compliance statements for MIB modules, which make use of
      the MODULE-COMPLIANCE and OBJECT-GROUP macros [2]; and,

(3)   capability statements for agent implementations which
      make use of the AGENT-CAPABILITIES macros [2].

This classification scheme does not imply a rigid taxonomy.
For example, a "standard" information module might include
definitions of managed objects and a compliance statement.
Similarly, an "enterprise-specific" information module might
include definitions of managed objects and a capability
statement. Of course, a "standard" information module may not
contain capability statements.

All information modules start with exactly one invocation of
the MODULE-IDENTITY macro, which provides contact and revision
history. This invocation must appear immediately after any
IMPORTs or EXPORTs statements.

3.1. Macro Invocation

Within an information module, each macro invocation appears
as:

        <descriptor> <macro> <clauses> ::= <value>

where <descriptor> corresponds to an ASN.1 identifier, <macro>

names the macro being invoked, and <clauses> and <value>
depend on the definition of the macro.

An ASN.1 identifier consists of one or more letters, digits,
or hyphens. The initial character must be a lower-case
letter, and the final character may not be a hyphen. Further,
a hyphen may not be immediatedly followed by another hyphen.

For all descriptors appearing in an information module, the
descriptor shall be unique and mnemonic, and shall not exceed
64 characters in length. This promotes a common language for
humans to use when discussing the information module and also
facilitates simple table mappings for user-interfaces.

The set of descriptors defined in all "standard" information
modules shall be unique. Further, within any information
module, the hyphen is not allowed as a character in any
descriptor.

Finally, by convention, if the descriptor refers to an object
with a SYNTAX clause value of either Counter32 or Counter64,
then the descriptor used for the object should denote
plurality.

3.1.1. Textual Clauses

Some clauses in a macro invocation may take a textual value
(e.g., the DESCRIPTION clause). Note that, in order to
conform to the ASN.1 syntax, the entire value of these clauses
must be enclosed in double quotation marks, and therefore
cannot itself contain double quotation marks, although the
value may be multi-line.

3.2. IMPORTing Symbols

To reference an external object, the IMPORTS statement must be
used to identify both the descriptor and the module defining
the descriptor.

Note that when symbols from "enterprise-specific" information
modules are referenced (e.g., a descriptor), there is the
possibility of collision. As such, if different objects with
the same descriptor are IMPORTed, then this ambiguity is

resolved by prefixing the descriptor with the name of the
information module and a dot ("."), i.e.,

    "module.descriptor"

(All descriptors must be unique within any information
module.)

Of course, this notation can be used even when there is no
collision when IMPORTing symbols.

Finally, the IMPORTS statement may not be used to import an
ASN.1 named type which corresponds to either the SEQUENCE or
SEQUENCE OF type.

4. Naming Hierarchy

The root of the subtree administered by the Internet Assigned
Numbers Authority (IANA) for the Internet is:

     internet   OBJECT IDENTIFIER ::= { iso 3 6 1 }

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

     1.3.6.1.

Several branches underneath this subtree are used for network
management:

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

However, the SMI does not prohibit the definition of objects
in other portions of the object tree.

The mgmt(2) subtree is used to identify "standard" objects.

The experimental(3) subtree is used to identify objects being
designed by working groups of the IETF. If an information
module produced by a working group becomes a "standard"
information module, then at the very beginning of its entry
onto the Internet standards track, the objects are moved under
the mgmt(2) subtree.

The private(4) subtree is used to identify objects defined
unilaterally. The enterprises(1) subtree beneath private is
used, among other things, to permit providers of networking
subsystems to register models of their products.

5. Mapping of the MODULE-IDENTITY macro

The MODULE-IDENTITY macro is used to provide contact and
revision history for each information module. It must appear
exactly once in every information module. It should be noted
that the expansion of the MODULE-IDENTITY macro is something
which conceptually happens during implementation and not
during run-time.

5.1. Mapping of the LAST-UPDATED clause

The LAST-UPDATED clause, which must be present, contains the
date and time that this information module was last edited.

5.2. Mapping of the ORGANIZATION clause

The ORGANIZATION clause, which must be present, contains a
textual description of the organization under whose auspices
this information module was developed.

5.3. Mapping of the CONTACT-INFO clause

The CONTACT-INFO clause, which must be present, contains the
name, postal address, telephone number, and electronic mail
address of the person to whom technical queries concerning
this information module should be sent.

5.4. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a
high-level textual description of the contents of this
information module.

5.5. Mapping of the REVISION clause

The REVISION clause, which need not be present, is repeatedly
used to describe the revisions made to this information
module, in reverse chronological order. Each instance of this
clause contains the date and time of the revision.

5.6. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present for each
REVISION clause, contains a high-level textual description of
the revision identified in that REVISION clause.

5.7. Mapping of the MODULE-IDENTITY value

The value of an invocation of the MODULE-IDENTITY macro is an
OBJECT IDENTIFIER. As such, this value may be authoritatively
used when referring to the information module containing the
invocation.

5.8. Usage Example

Consider how a skeletal MIB module might be constructed: e.g.,

```
FIZBIN-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, experimental
        FROM SNMPv2-SMI;


fizbin MODULE-IDENTITY
    LAST-UPDATED "9210070433Z"
    ORGANIZATION "IETF SNMPv2 Working Group"
    CONTACT-INFO
            "          Marshall T. Rose

              Postal: Dover Beach Consulting, Inc.
                      420 Whisman Court
                      Mountain View, CA  94043-2186
                      US

                 Tel: +1 415 968 1052
                 Fax: +1 415 968 2510

              E-mail: mrose@dbc.mtview.ca.us"
    DESCRIPTION
            "The MIB module for entities implementing the xxxx
            protocol."
    REVISION        "9210070433Z"
    DESCRIPTION
            "Initial version of this MIB module."
-- contact IANA for actual number
    ::= { experimental xx }


END
```

6. Mapping of the OBJECT-IDENTITY macro

The OBJECT-IDENTITY macro is used to define information about
an OBJECT IDENTIFIER assignment. It should be noted that the
expansion of the OBJECT-IDENTITY macro is something which
conceptually happens during implementation and not during
run-time.

6.1. Mapping of the STATUS clause

The STATUS clause, which must be present, indicates whether
this definition is current or historic.

The values "current", and "obsolete" are self-explanatory.

6.2. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a
textual description of the object assignment.

6.3. Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a
textual cross-reference to an object assignment defined in
some other information module.

6.4. Mapping of the OBJECT-IDENTITY value

The value of an invocation of the OBJECT-IDENTITY macro is an
OBJECT IDENTIFIER.

6.5. Usage Example

Consider how an OBJECT IDENTIFIER assignment might be made:
e.g.,

```
fizbin69 OBJECT-IDENTITY
    STATUS  current
    DESCRIPTION
            "The authoritative identity of the Fizbin 69
            chipset."
    ::= { fizbinChipSets 1 }
```

7. Mapping of the OBJECT-TYPE macro

The OBJECT-TYPE macro is used to define a managed object. It
should be noted that the expansion of the OBJECT-TYPE macro is
something which conceptually happens during implementation and
not during run-time.

7.1. Mapping of the SYNTAX clause

The SYNTAX clause, which must be present, defines the abstract
data structure corresponding to that object. The data
structure must be one of the alternatives defined in the
ObjectSyntax CHOICE.

Full ASN.1 sub-typing is allowed, as appropriate to the
underingly ASN.1 type, primarily as an aid to implementors in
understanding the meaning of the object. Any such restriction
on size, range, enumerations or repertoire specified in this
clause represents the maximal level of support which makes
"protocol sense". Of course, sub-typing is not allowed for
the Counter32 or Counter64 types, but is allowed for the
Gauge32 type.

The semantics of ObjectSyntax are now described.

7.1.1. Integer32 and INTEGER

The Integer32 type represents integer-valued information
between -2^{31} and $2^{31}-1$ inclusive (-2147483648 to 2147483647
decimal). This type is indistinguishable from the INTEGER
type.

The INTEGER type may also be used to represent integer-valued
information, if it contains named-number enumerations, or if
it is sub-typed to be more constrained than the Integer32
type. In the former case, only those named-numbers so
enumerated may be present as a value. Note that although it
is recommended that enumerated values start at 1 and be
numbered contiguously, any valid value for Integer32 is
allowed for an enumerated value and, further, enumerated
values needn't be contiguously assigned.

Finally, the hyphen character is not allowed as a part of the
label name for any named-number enumeration.

7.1.2. OCTET STRING

The OCTET STRING type represents arbitrary binary or textual
data. Although there is no SMI-specified size limitation for
this type, MIB designers should realize that there may be
implementation and interoperability limitations for sizes in
excess of 255 octets.

7.1.3. OBJECT IDENTIFIER

The OBJECT IDENTIFIER type represents administratively
assigned names. Any instance of this type may have at most
128 sub-identifiers. Further, each sub-identifier must not
exceed the value 2^32-1 (4294967295 decimal).

7.1.4. BIT STRING

The BIT STRING type represents an enumeration of named bits.
This collection is assigned non-negative, contiguous values,
starting at zero. Only those named-bits so enumerated may be
present in a value.

A requirement on "standard" MIB modules is that the hyphen
character is not allowed as a part of the label name for any
named-bit enumeration.

7.1.5. IpAddress

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

Note that the IpAddress type is a tagged type for historical
reasons. Network addresses should be represented using an
invocation of the TEXTUAL-CONVENTION macro [3].

7.1.6. Counter32

The Counter32 type represents a non-negative integer which
monotonically increases until it reaches a maximum value of
2^32-1 (4294967295 decimal), when it wraps around and starts
increasing again from zero.

Counters have no defined "initial" value, and thus, a single
value of a Counter has (in general) no information content.
Discontinuities in the monotonically increasing value normally
occur at re-initialization of the management system, and at
other times as specified in the description of an object-type
using this ASN.1 type. If such other times can occur, for
example, the creation of an object instance at times other
than re-initialization, then a corresponding object should be
defined with a SYNTAX clause value of TimeStamp (a textual
convention defined in [3]) indicating the time of the last
discontinuity.

The value of the MAX-ACCESS clause for objects with a SYNTAX
clause value of Counter32 is always "read-only".

A DEFVAL clause is not allowed for objects with a SYNTAX
clause value of Counter32.

7.1.7. Gauge32

The Gauge32 type represents a non-negative integer, which may
increase or decrease, but shall never exceed a maximum value.
The maximum value can not be greater than 2^32-1 (4294967295
decimal). The value of a Gauge has its maximum value whenever
the information being modeled is greater or equal to that
maximum value; if the information being modeled subsequently
decreases below the maximum value, the Gauge also decreases.

7.1.8. TimeTicks

The TimeTicks type represents a non-negative integer which
represents the time, modulo 2^32 (4294967296 decimal), in
hundredths of a second between two epochs. When objects are
defined which use this ASN.1 type, the description of the
object identifies both of the reference epochs.

For example, [3] defines the TimeStamp textual convention
which is based on the TimeTicks type. With a TimeStamp, the
first reference epoch is defined as when MIB-II's sysUpTime
[7] was zero, and the second reference epoch is defined as the
current value of sysUpTime.

7.1.9. Opaque

The Opaque type is provided solely for backward-compatibility,
and shall not be used for newly-defined object types.

The Opaque type supports the capability to pass arbitrary
ASN.1 syntax. A value is encoded using the ASN.1 Basic
Encoding Rules [4] 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.

A requirement on "standard" MIB modules is that no object may
have a SYNTAX clause value of Opaque.

7.1.10. NsapAddress

The NsapAddress type represents an OSI address as a variable-
length OCTET STRING. The first octet of the string contains a
binary value in the range of 0..20, and indicates the length
in octets of the NSAP. Following the first octet, is the
NSAP, expressed in concrete binary notation, starting with the
most significant octet. A zero-length NSAP is used as a
"special" address meaning "the default NSAP" (analogous to the
IP address of 0.0.0.0). Such an NSAP is encoded as a single
octet, containing the value 0. All other NSAPs are encoded in
at least 4 octets.

Note that the NsapAddress type is a tagged type for historical
reasons. Network addresses should be represented using an
invocation of the TEXTUAL-CONVENTION macro [3].

7.1.11. Counter64

The Counter64 type represents a non-negative integer which monotonically increases until it reaches a maximum value of 2^64-1 (18446744073709551615 decimal), when it wraps around and starts increasing again from zero.

Counters have no defined "initial" value, and thus, a single value of a Counter has (in general) no information content. Discontinuities in the monotonically increasing value normally occur at re-initialization of the management system, and at other times as specified in the description of an object-type using this ASN.1 type. If such other times can occur, for example, the creation of an object instance at times other than re-initialization, then a corresponding object should be defined with a SYNTAX clause value of TimeStamp (a textual convention defined in [3]) indicating the time of the last discontinuity.

The value of the MAX-ACCESS clause for objects with a SYNTAX clause value of Counter64 is always "read-only".

A requirement on "standard" MIB modules is that the Counter64 type may be used only if the information being modeled would wrap in less than one hour if the Counter32 type was used instead.

A DEFVAL clause is not allowed for objects with a SYNTAX clause value of Counter64.

7.1.12. UInteger32

The UInteger32 type represents integer-valued information between 0 and 2^32-1 inclusive (0 to 4294967295 decimal).

7.2. Mapping of the UNITS clause

This UNITS clause, which need not be present, contains a textual definition of the units associated with that object.

7.3. Mapping of the MAX-ACCESS clause

The MAX-ACCESS clause, which must be present, defines whether it makes "protocol sense" to read, write and/or create an instance of the object. This is the maximal level of access for the object. (This maximal level of access is independent of any administrative authorization policy.)

The value "read-write" indicates that read and write access make "protocol sense", but create does not. The value "read-create" indicates that read, write and create access make "protocol sense". The value "not-accessible" indicates either an auxiliary object (see Section 7.7) or an object which is accessible only via a notificationn (e.g., snmpTrapOID [5]).

These values are ordered, from least to greatest: "not-accessible", "read-only", "read-write", "read-create".

If any columnar object in a conceptual row has "read-create" as its maximal level of access, then no other columnar object of the same conceptual row may have a maximal access of "read-write". (Note that "read-create" is a superset of "read-write".)

7.4. Mapping of the STATUS clause

The STATUS clause, which must be present, indicates whether this definition is current or historic.

The values "current", and "obsolete" are self-explanatory. The "deprecated" value indicates that the object is obsolete, but that an implementor may wish to support that object to foster interoperability with older implementations.

7.5. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a textual definition of that object 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.

7.6. 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
information module. This is useful when de-osifying a MIB
module produced by some other organization.

7.7. Mapping of the INDEX clause

The INDEX clause, which must be present if that object
corresponds to a conceptual row (unless an AUGMENTS clause is
present instead), and must be absent otherwise, defines
instance identification information for the columnar objects
subordinate to that object.

Management 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. This conceptualization is 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. A conceptual table has SYNTAX of the form:

        SEQUENCE OF <EntryType>

where <EntryType> refers to the SEQUENCE type of its
subordinate conceptual row. A conceptual row has SYNTAX of
the form:

        <EntryType>

where <EntryType> is a SEQUENCE type defined as follows:

        <EntryType> ::= SEQUENCE { <type1>, ... , <typeN> }

where there is one <type> for each subordinate object, and
each <type> is of the form:

        <descriptor> <syntax>

where <descriptor> is the descriptor naming a subordinate

object, and <syntax> has the value of that subordinate
object's SYNTAX clause, optionally omitting the sub-typing
information. Further, these ASN.1 types are always present
(the DEFAULT and OPTIONAL clauses are disallowed in the
SEQUENCE definition). The MAX-ACCESS clause for conceptual
tables and rows is "not-accessible".

For leaf objects which are not columnar objects, instances of
the object are identified by appending a sub-identifier of
zero to the name of that object. Otherwise, the INDEX clause
of the conceptual row object superior to a columnar object
defines instance identification information.

The instance identification information in an INDEX clause
must specify object(s) such that value(s) of those object(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 (or variable-length
      preceded by the IMPLIED keyword): `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 (not preceded by
      the IMPLIED keyword): `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)   IpAddress-valued: 4 sub-identifiers, in the familiar
      a.b.c.d notation.

(6)   NsapAddress-valued: `n' sub-identifiers, where `n' is the
      length of the value (each octet of the value is encoded
      in a separate sub-identifier);

Note that the IMPLIED keyword can only be present for objects
having a variable-length syntax (e.g., variable-length strings
or object identifier-valued objects). Further, the IMPLIED
keyword may appear at most once within the INDEX clause, and
if so, is associated with the right-most object having a
variable-length syntax. Finally, the IMPLIED keyword may not
be used on a variable-length string object if that string
might have a value of zero-length.

Instances identified by use of integer-valued objects should
be numbered starting from one (i.e., not from zero). The use
of zero as a value for an integer-valued index object should
be avoided, except in special cases.

Objects which are both specified in the INDEX clause of a
conceptual row and also columnar objects of the same
conceptual row are termed auxiliary objects. The MAX-ACCESS
clause for newly-defined auxiliary objects is "not-
accessible". However, a conceptual row must contain at least
one columnar object which is not an auxiliary object (i.e.,
the value of the MAX-ACCESS clause for such an object is
either "read-only" or "read-create").

Note that objects specified in a conceptual row's INDEX clause
need not be columnar objects of that conceptual row. In this
situation, the DESCRIPTION clause of the conceptual row must
include a textual explanation of how the objects which are
included in the INDEX clause but not columnar objects of that
conceptual row, are used in uniquely identifying instances of
the conceptual row's columnar objects.

7.7.1. Creation and Deletion of Conceptual Rows

For newly-defined conceptual rows which allow the creation of
new object instances and the deletion of existing object
instances, there should be one columnar object with a SYNTAX
clause value of RowStatus (a textual convention defined in
[3]) and a MAX-ACCESS clause value of read-create. By
convention, this is termed the status column for the
conceptual row.

7.8. Mapping of the AUGMENTS clause

The AUGMENTS clause, which must not be present unless the
object corresponds to a conceptual row, is an alternative to
the INDEX clause. Every object corresponding to a conceptual
row has either an INDEX clause or an AUGMENTS clause.

If an object corresponding to a conceptual row has an INDEX
clause, that row is termed a base conceptual row;
alternatively, if the object has an AUGMENTS clause, the row
is said to be a conceptual row augmentation, where the
AUGMENTS clause names the object corresponding to the base
conceptual row which is augmented by this conceptual row
extension. Instances of subordinate columnar objects of a
conceptual row extension are identified according to the INDEX
clause of the base conceptual row corresponding to the object
named in the AUGMENTS clause. Further, instances of
subordinate columnar objects of a conceptual row extension
exist according to the same semantics as instances of
subordinate columnar objects of the base conceptual row being
augmented. As such, note that creation of a base conceptual
row implies the correspondent creation of any conceptual row
augmentations.

For example, a MIB designer might wish to define additional
columns in an "enterprise-specific" MIB which logically extend
a conceptual row in a "standard" MIB. The "standard" MIB
definition of the conceptual row would include the INDEX
clause and the "enterprise-specific" MIB would contain the
definition of a conceptual row using the AUGMENTS clause.

Note that a base conceptual row may be augmented by multiple
conceptual row extensions.

7.8.1. Relation between INDEX and AUGMENTS clauses

When defining instance identification information for a
conceptual table:

(1) If there is a one-to-one correspondence between the
     conceptual rows of this table and an existing table, then
     the AUGMENTS clause should be used.

(2)   Otherwise, if there is a sparse relationship between the
        conceptuals rows of this table and an existing table,
        then an INDEX clause should be used which is identical to
        that in the existing table.

(3)   Otherwise, auxiliary objects should be defined within the
        conceptual row for the new table, and those objects
        should be used within the INDEX clause for the conceptual
        row.

7.9. Mapping of the DEFVAL clause

The DEFVAL clause, which need not be present, defines an
acceptable default value which may be used at the discretion
of a SNMPv2 entity acting in an agent role when an object
instance is created.

During conceptual row creation, if an instance of a columnar
object is not present as one of the operands in the
correspondent management protocol set operation, then the
value of the DEFVAL clause, if present, indicates an
acceptable default value that a SNMPv2 entity acting in an
agent role might use.

The value of the DEFVAL clause must, of course, correspond to
the SYNTAX clause for the object. If the value is an OBJECT
IDENTIFIER, then it must be expressed as a single ASN.1
identifier, and not as a collection of sub-identifiers.

Note that if an operand to the management protocol set
operation is an instance of a read-only object, then the error
`notWritable' [6] will be returned. As such, the DEFVAL
clause can be used to provide an acceptable default value that
a SNMPv2 entity acting in an agent role might use.

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

```
ObjectSyntax          DEFVAL clause
-----------------     ------------
Integer32             1
                      -- same for Gauge32, TimeTicks, UInteger32
INTEGER               valid -- enumerated value
OCTET STRING          'ffffffffffff'H
OBJECT IDENTIFIER     sysDescr
BIT STRING            { primary, secondary } -- enumerated values
     IpAddress            'c0210415'H -- 192.33.4.21
```

Object types with SYNTAX of Counter32 and Counter64 may not
have DEFVAL clauses, since they do not have defined initial
values. However, it is recommended that they be initialized
to zero.

7.10. 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, an
administratively assigned name.

When an OBJECT IDENTIFIER is assigned to an object:

(1) If the object corresponds to a conceptual table, then
     only a single assignment, that for a conceptual row, is
     present immediately beneath that object. The
     administratively assigned name for the conceptual row
     object is derived by appending a sub-identifier of "1" to
     the administratively assigned name for the conceptual
     table.

(2) If the object corresponds to a conceptual row, then at
     least one assignment, one for each column in the
     conceptual row, is present beneath that object. The
     administratively assigned name for each column is derived
     by appending a unique, positive sub-identifier to the
     administratively assigned name for the conceptual row.

(3) Otherwise, no other OBJECT IDENTIFIERs which are
     subordinate to the object may be assigned.

Note that the final sub-identifier of any administratively
assigned name for an object shall be positive. A zero-valued
final sub-identifier is reserved for future use.

Further note that although conceptual tables and rows are given administratively assigned names, these conceptual objects may not be manipulated in aggregate form by the management protocol.

7.11. Usage Example

Consider how one might define a conceptual table and its
subordinates.

```
evalSlot OBJECT-TYPE
    SYNTAX      INTEGER
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The index number of the first unassigned entry in
            the evaluation table.

            A management station should create new entries in
            the evaluation table using this algorithm: first,
            issue a management protocol retrieval operation to
            determine the value of evalSlot; and, second,
            issue a management protocol set operation to
            create an instance of the evalStatus object
            setting its value to underCreation(1).  If this
            latter operation succeeds, then the management
            station may continue modifying the instances
            corresponding to the newly created conceptual row,
            without fear of collision with other management
            stations."
    ::= { eval 1 }

evalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EvalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The (conceptual) evaluation table."
    ::= { eval 2 }

evalEntry OBJECT-TYPE
    SYNTAX      EvalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "An entry (conceptual row) in the evaluation
            table."
    INDEX   { evalIndex }
    ::= { evalTable 1 }
```

```
EvalEntry ::=
    SEQUENCE {
        evalIndex        Integer32,
        evalString       DisplayString,
        evalValue        Integer32,
        evalStatus       RowStatus
    }

evalIndex OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
            "The auxiliary variable used for identifying
            instances of the columnar objects in the
            evaluation table."
        ::= { evalEntry 1 }

evalString OBJECT-TYPE
    SYNTAX       DisplayString
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
            "The string to evaluate."
        ::= { evalEntry 2 }

evalValue OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
            "The value when evalString was last executed."
    DEFVAL  { 0 }
        ::= { evalEntry 3 }

evalStatus OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
            "The status column used for creating, modifying,
            and deleting instances of the columnar objects in
            the evaluation  table."
    DEFVAL  { active }
        ::= { evalEntry 4 }
```

8. Mapping of the NOTIFICATION-TYPE macro

The NOTIFICATION-TYPE macro is used to define the information
contained within an unsolicited transmission of management
information (i.e., within either a SNMPv2-Trap-PDU or
InformRequest-PDU). It should be noted that the expansion of
the NOTIFICATION-TYPE macro is something which conceptually
happens during implementation and not during run-time.

8.1. Mapping of the OBJECTS clause

The OBJECTS clause, which need not be present, defines the
ordered sequence of MIB objects which are contained within
every instance of the notification.

8.2. Mapping of the STATUS clause

The STATUS clause, which must be present, indicates whether
this definition is current or historic.

The values "current", and "obsolete" are self-explanatory.
The "deprecated" value indicates that the notification is
obsolete, but that an implementor may wish to support that
object to foster interoperability with older implementations.

8.3. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a
textual definition of the notification 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. In particular, the DESCRIPTION clause should document
which instances of the objects mentioned in the OBJECTS clause
should be contained within notifications of this type.

8.4. Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a
textual cross-reference to a notification defined in some
other information module. This is useful when de-osifying a

MIB module produced by some other organization.

8.5. Mapping of the NOTIFICATION-TYPE value

The value of an invocation of the NOTIFICATION-TYPE macro is
the name of the notification, which is an OBJECT IDENTIFIER,
an administratively assigned name.

Sections 4.2.6 and 4.2.7 of [6] describe how the
NOTIFICATION-TYPE macro is used to generate a SNMPv2-Trap-PDU
or InformRequest-PDU, respectively.

8.6. Usage Example

Consider how a linkUp trap might be described:

```
linkUp NOTIFICATION-TYPE
    OBJECTS { ifIndex }
    STATUS  current
    DESCRIPTION
            "A linkUp trap signifies that the SNMPv2 entity,
            acting in an agent role, recognizes that one of
            the communication links represented in its
            configuration has come up."
    ::= { snmpTraps 4 }
```

According to this invocation, the trap authoritatively
identified as

    { snmpTraps 4 }

is used to report a link coming up.

Note that a SNMPv2 entity acting in an agent role can be
configured to send this trap to zero or more SNMPv2 entities
acting in a manager role, depending on the contents of the
aclTable and viewTable [8] tables. For example, by judicious
use of the viewTable, a SNMPv2 entity acting in an agent role
might be configured to send all linkUp traps to one particular
SNMPv2 entity, and linkUp traps for only certain interfaces to
other SNMPv2 entities.

9. Refined Syntax

Some macros allow an object's syntax to be refined (e.g., the
SYNTAX clause in the MODULE-COMPLIANCE macro [2]). However,
not all refinements of syntax are appropriate. In particular,
the object's primitive or application type must not be
changed.

Further, the following restrictions apply:

	Restrictions to Refinement on			
object syntax	range	enumeration	size	repertoire
INTEGER	(1)	(2)	-	-
OCTET STRING	-	-	(3)	(4)
OBJECT IDENTIFIER	-	-	-	-
BIT STRING	-	(2)	-	-
IpAddress	-	-	-	-
Counter32	-	-	-	-
Gauge32	(1)	-	-	-
TimeTicks	-	-	-	-
NsapAddress	-	-	-	-
Counter64	-	-	-	-

where:

(1)   the range of permitted values may be refined by raising
      the lower-bounds, by reducing the upper-bounds, and/or by
      reducing the alternative value/range choices;

(2)   the enumeration of named-values may be refined by
      removing one or more named-values;

(3)   the size in characters of the value may be refined by
      raising the lower-bounds, by reducing the upper-bounds,
      and/or by reducing the alternative size choices; or,

(4)   the repertoire of characters in the value may be reduced
      by further sub-typing.

Otherwise no refinements are possible.

Note that when refining an object with a SYNTAX clause value
of Integer32 or UInteger32, the refined SYNTAX is expressed as
an INTEGER and the restrictions of the table above are used.

10. Extending an Information Module

As experience is gained with a published information module,
it may be desirable to revise that information module.

To begin, the invocation of the MODULE-IDENTITY macro should
be updated to include information about the revision.
Usually, this consists of updating the LAST-UPDATED clause and
adding a pair of REVISION and DESCRIPTION clauses. However,
other existing clauses in the invocation may be updated.

Note that the module's label (e.g., "FIZBIN-MIB" from the
example in Section 5.8), is not changed when the information
module is revised.

10.1. Object Assignments

If any non-editorial change is made to any clause of a object
assignment, then the OBJECT IDENTIFIER value associated with
that object assignment must also be changed, along with its
associated descriptor.

10.2. Object Definitions

An object definition may be revised in any of the following
ways:

(1) A SYNTAX clause containing an enumerated INTEGER may have
     new enumerations added or existing labels changed.

(2) A STATUS clause value of "current" may be revised as
     "deprecated" or "obsolete". Similarly, a STATUS clause
     value of "deprecated" may be revised as "obsolete".

(3) A DEFVAL clause may be added or updated.

(4) A REFERENCE clause may be added or updated.

(5) A UNITS clause may be added.

(6) A conceptual row may be augmented by adding new columnar
     objects at the end of the row.

(7) Entirely new objects may be defined, named with
     previously unassigned OBJECT IDENTIFIER values.

Otherwise, if the semantics of any previously defined object
are changed (i.e., if a non-editorial change is made to any
clause other those specifically allowed above), then the
OBJECT IDENTIFIER value associated with that object must also
be changed.

Note that changing the descriptor associated with an existing
object is considered a semantic change, as these strings may
be used in an IMPORTS statement.

Finally, note that if an object has the value of its STATUS
clause changed, then the value of its DESCRIPTION clause
should be updated accordingly.

10.3. Notification Definitions

A notification definition may be revised in any of the
following ways:

(1) A REFERENCE clause may be added or updated.

Otherwise, if the semantics of any previously defined
notification are changed (i.e., if a non-editorial change is
made to any clause other those specifically allowed above),
then the OBJECT IDENTIFIER value associated with that
notification must also be changed.

Note that changing the descriptor associated with an existing
notification is considered a semantic change, as these strings
may be used in an IMPORTS statement.

Finally, note that if an object has the value of its STATUS
clause changed, then the value of its DESCRIPTION clause
should be updated accordingly.

11. Appendix: de-OSIfying a MIB module

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, as shown
earlier in Section 5.8. The next step is to categorize the
objects into groups. Optional objects are not permitted.
Thus, when a MIB module is created, optional objects must be
placed in a additional groups, 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.

11.1. 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 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 MIBs with a myriad of optionals.

11.1.1. Mapping to the SYNTAX clause

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

(1)   An object with BOOLEAN syntax becomes a TruthValue [3].

(2)   An object with INTEGER syntax becomes an Integer32.

(3)   An object with ENUMERATED syntax becomes an INTEGER with
      enumerations, taking any of the values given which can be
      represented with an Integer32.

(4)   An object with BIT STRING syntax but no enumerations
      becomes an OCTET STRING.

(5)   An object with a character string syntax becomes either
      an OCTET STRING, or a DisplayString [3], depending on the
      repertoire of the character string.

(6)   A 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.

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

11.1.2. Mapping to the UNITS clause

If the description of this managed object defines a unit-
basis, then mapping to this clause is straight-forward.

11.1.3. Mapping to the MAX-ACCESS clause

This is straight-forward.

11.1.4. Mapping to the STATUS clause

This is straight-forward.

11.1.5. 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).

11.1.6. 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.

11.1.7. Mapping to the INDEX clause

If necessary, decide how instance-identifiers for columnar
objects are to be formed and define this clause accordingly.

11.1.8. 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.

11.2. Action Mapping

Actions are modeled as read-write objects, in which writing a
particular value results in a state change. (Usually, as a
part of this state change, some action might take place.)

11.2.1. Mapping to the SYNTAX clause

Usually the Integer32 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.

11.2.2. Mapping to the MAX-ACCESS clause

Always use read-write or read-create.

11.2.3. Mapping to the STATUS clause

This is straight-forward.

11.2.4. 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).

11.2.5. 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.

11.3. Event Mapping

Events are modeled as SNMPv2 notifications using
NOTIFICATION-TYPE macro. However, recall that SNMPv2
emphasizes trap-directed polling. As such, few, and usually
no, notifications, need be defined for any MIB module.

11.3.1. Mapping to the STATUS clause

This is straight-forward.

11.3.2. 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).

11.3.3. Mapping to the REFERENCE clause

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

12. Acknowledgements

The section on object definitions (and MIB de-osification) is based, in part, on RFCs 1155 and 1212. The IMPLIED keyword is based on a conversation with David T. Perkins in December, 1991.

The section on trap definitions is based, in part, on RFC 1215.

Finally, the comments of the SNMP version 2 working group are gratefully acknowledged:

        Beth Adams, Network Management Forum
        Steve Alexander, INTERACTIVE Systems Corporation
        David Arneson, Cabletron Systems
        Toshiya Asaba
        Fred Baker, ACC
        Jim Barnes, Xylogics, Inc.
        Brian Bataille
        Andy Bierman, SynOptics Communications, Inc.
        Uri Blumenthal, IBM Corporation
        Fred Bohle, Interlink
        Jack Brown
        Theodore Brunner, Bellcore
        Stephen F. Bush, GE Information Services
        Jeffrey D. Case, University of Tennessee, Knoxville
        John Chang, IBM Corporation
        Szusin Chen, Sun Microsystems
        Robert Ching
        Chris Chiotasso, Ungermann-Bass
        Bobby A. Clay, NASA/Boeing
        John Cooke, Chipcom
        Tracy Cox, Bellcore
        Juan Cruz, Datability, Inc.
        David Cullerot, Cabletron Systems
        Cathy Cunningham, Microcom
        James R. (Chuck) Davin, Bellcore
        Michael Davis, Clearpoint
        Mike Davison, FiberCom
        Cynthia DellaTorre, MITRE
        Taso N. Devetzis, Bellcore
        Manual Diaz, DAVID Systems, Inc.
        Jon Dreyer, Sun Microsystems
        David Engel, Optical Data Systems

Mike Erlinger, Lexcel
Roger Fajman, NIH
Daniel Fauvarque, Sun Microsystems
Karen Frisa, CMU
Shari Galitzer, MITRE
Shawn Gallagher, Digital Equipment Corporation
Richard Graveman, Bellcore
Maria Greene, Xyplex, Inc.
Michel Guittet, Apple
Robert Gutierrez, NASA
Bill Hagerty, Cabletron Systems
Gary W. Haney, Martin Marietta Energy Systems
Patrick Hanil, Nokia Telecommunications
Matt Hecht, SNMP Research, Inc.
Edward A. Heiner, Jr., Synernetics Inc.
Susan E. Hicks, Martin Marietta Energy Systems
Geral Holzhauer, Apple
John Hopprich, DAVID Systems, Inc.
Jeff Hughes, Hewlett-Packard
Robin Iddon, Axon Networks, Inc.
David Itusak
Kevin M. Jackson, Concord Communications, Inc.
Ole J. Jacobsen, Interop Company
Ronald Jacoby, Silicon Graphics, Inc.
Satish Joshi, SynOptics Communications, Inc.
Frank Kastenholz, FTP Software
Mark Kepke, Hewlett-Packard
Ken Key, SNMP Research, Inc.
Zbiginew Kielczewski, Eicon
Jongyeoi Kim
Andrew Knutsen, The Santa Cruz Operation
Michael L. Kornegay, VisiSoft
Deirdre C. Kostik, Bellcore
Cheryl Krupczak, Georgia Tech
Mark S. Lewis, Telebit
David Lin
David Lindemulder, AT&T/NCR
Ben Lisowski, Sprint
David Liu, Bell-Northern Research
John Lunny, The Wollongong Group
Robert C. Lushbaugh Martin, Marietta Energy Systems
Michael Luufer, BBN
Carl Madison, Star-Tek, Inc.
Keith McCloghrie, Hughes LAN Systems
Evan McGinnis, 3Com Corporation

Bill McKenzie, IBM Corporation
Donna McMaster, SynOptics Communications, Inc.
John Medicke, IBM Corporation
Doug Miller, Telebit
Dave Minnich, FiberCom
Mohammad Mirhakkak, MITRE
Rohit Mital, Protools
George Mouradian, AT&T Bell Labs
Patrick Mullaney, Cabletron Systems
Dan Myers, 3Com Corporation
Rina Nathaniel, Rad Network Devices Ltd.
Hien V. Nguyen, Sprint
Mo Nikain
Tom Nisbet
William B. Norton, MERIT
Steve Onishi, Wellfleet Communications, Inc.
David T. Perkins, SynOptics Communications, Inc.
Carl Powell, BBN
Ilan Raab, SynOptics Communications, Inc.
Richard Ramons, AT&T
Venkat D. Rangan, Metric Network Systems, Inc.
Louise Reingold, Sprint
Sam Roberts, Farallon Computing, Inc.
Kary Robertson, Concord Communications, Inc.
Dan Romascanu, Lannet Data Communications Ltd.
Marshall T. Rose, Dover Beach Consulting, Inc.
Shawn A. Routhier, Epilogue Technology Corporation
Chris Rozman
Asaf Rubissa, Fibronics
Jon Saperia, Digital Equipment Corporation
Michael Sapich
Mike Scanlon, Interlan
Sam Schaen, MITRE
John Seligson, Ultra Network Technologies
Paul A. Serice, Corporation for Open Systems
Chris Shaw, Banyan Systems
Timon Sloane
Robert Snyder, Cisco Systems
Joo Young Song
Roy Spitier, Sprint
Einar Stefferud, Network Management Associates
John Stephens, Cayman Systems, Inc.
Robert L. Stewart, Xyplex, Inc. (chair)
Kaj Tesink, Bellcore
Dean Throop, Data General

13. 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] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S.,
     "Conformance Statements for version 2 of the the Simple
     Network Management Protocol (SNMPv2)", RFC 1444, SNMP
     Research, Inc., Hughes LAN Systems, Dover Beach
     Consulting, Inc., Carnegie Mellon University, April 1993.

[3] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S.,
     "Textual Conventions for version 2 of the the Simple
     Network Management Protocol (SNMPv2)", RFC 1443, SNMP
     Research, Inc., Hughes LAN Systems, Dover Beach
     Consulting, Inc., Carnegie Mellon University, April 1993.

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

[5] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S.,
     "Management Information Base for version 2 of the Simple
     Network Management Protocol (SNMPv2)", RFC 1450, SNMP
     Research, Inc., Hughes LAN Systems, Dover Beach
     Consulting, Inc., Carnegie Mellon University, April 1993.

[6] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S.,
     "Protocol Operations for version 2 of the Simple Network
     Management Protocol (SNMPv2)", RFC 1448, SNMP Research,
     Inc., Hughes LAN Systems, Dover Beach Consulting, Inc.,
     Carnegie Mellon University, April 1993.

[7] McCloghrie, K., and Rose, M., "Management Information
     Base for Network Management of TCP/IP-based internets:
     MIB-II", STD 17, RFC 1213, March 1991.

[8] McCloghrie, K., and Galvin, J., "Party MIB for version 2
     of the Simple Network Management Protocol (SNMPv2)", RFC
     1447, Hughes LAN Systems, Trusted Information Systems,

April 1993.

14. Security Considerations

Security issues are not discussed in this memo.

15.   Authors' Addresses

Jeffrey D. Case
SNMP Research, Inc.
3001 Kimberlin Heights Rd.
Knoxville, TN 37920-9716
US

Phone: +1 615 573 1434
Email: case@snmp.com

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

Phone: +1 415 966 7934
Email: kzm@hls.com

Marshall T. Rose
Dover Beach Consulting, Inc.
420 Whisman Court
Mountain View, CA 94043-2186
US

Phone: +1 415 968 1052
Email: mrose@dbc.mtview.ca.us

Steven Waldbusser
Carnegie Mellon University
4910 Forbes Ave
Pittsburgh, PA 15213
US

Phone: +1 412 268 6628
Email: waldbusser@cmu.edu