

Network Working Group
Request for Comments: 2578
STD: 58
Obsoletes: 1902
Category: Standards Track

Editors of this version:
K. McCloghrie
Cisco Systems
D. Perkins
SNMPinfo
J. Schoenwaelder
TU Braunschweig
Authors of previous version:
J. Case
SNMP Research
K. McCloghrie
Cisco Systems
M. Rose
First Virtual Holdings
S. Waldbusser
International Network Services
April 1999

Structure of Management Information Version 2 (SMIv2)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Table of Contents

1 Introduction	3
1.1 A Note on Terminology	4
2 Definitions	4
2.1 The MODULE-IDENTITY macro	5
2.2 Object Names and Syntaxes	5
2.3 The OBJECT-TYPE macro	8
2.5 The NOTIFICATION-TYPE macro	10
2.6 Administrative Identifiers	11
3 Information Modules	11
3.1 Macro Invocation	12
3.1.1 Textual Values and Strings	13

3.2	IMPORTing Symbols	14
3.3	Exporting Symbols	14
3.4	ASN.1 Comments	14
3.5	OBJECT IDENTIFIER values	15
3.6	OBJECT IDENTIFIER usage	15
3.7	Reserved Keywords	16
4	Naming Hierarchy	16
5	Mapping of the MODULE-IDENTITY macro	17
5.1	Mapping of the LAST-UPDATED clause	17
5.2	Mapping of the ORGANIZATION clause	17
5.3	Mapping of the CONTACT-INFO clause	18
5.4	Mapping of the DESCRIPTION clause	18
5.5	Mapping of the REVISION clause	18
5.5.1	Mapping of the DESCRIPTION sub-clause	18
5.6	Mapping of the MODULE-IDENTITY value	18
5.7	Usage Example	18
6	Mapping of the OBJECT-IDENTITY macro	19
6.1	Mapping of the STATUS clause	19
6.2	Mapping of the DESCRIPTION clause	20
6.3	Mapping of the REFERENCE clause	20
6.4	Mapping of the OBJECT-IDENTITY value	20
6.5	Usage Example	20
7	Mapping of the OBJECT-TYPE macro	20
7.1	Mapping of the SYNTAX clause	21
7.1.1	Integer32 and INTEGER	21
7.1.2	OCTET STRING	21
7.1.3	OBJECT IDENTIFIER	22
7.1.4	The BITS construct	22
7.1.5	IpAddress	22
7.1.6	Counter32	23
7.1.7	Gauge32	23
7.1.8	TimeTicks	24
7.1.9	Opaque	24
7.1.10	Counter64	24
7.1.11	Unsigned32	25
7.1.12	Conceptual Tables	25
7.1.12.1	Creation and Deletion of Conceptual Rows	26
7.2	Mapping of the UNITS clause	26
7.3	Mapping of the MAX-ACCESS clause	26
7.4	Mapping of the STATUS clause	27
7.5	Mapping of the DESCRIPTION clause	27
7.6	Mapping of the REFERENCE clause	27
7.7	Mapping of the INDEX clause	27
7.8	Mapping of the AUGMENTS clause	29
7.8.1	Relation between INDEX and AUGMENTS clauses	30
7.9	Mapping of the DEFVAL clause	30
7.10	Mapping of the OBJECT-TYPE value	31
7.11	Usage Example	32

8 Mapping of the NOTIFICATION-TYPE macro	34
8.1 Mapping of the OBJECTS clause	34
8.2 Mapping of the STATUS clause	34
8.3 Mapping of the DESCRIPTION clause	35
8.4 Mapping of the REFERENCE clause	35
8.5 Mapping of the NOTIFICATION-TYPE value	35
8.6 Usage Example	35
9 Refined Syntax	36
10 Extending an Information Module	37
10.1 Object Assignments	37
10.2 Object Definitions	38
10.3 Notification Definitions	39
11 Appendix A: Detailed Sub-typing Rules	40
11.1 Syntax Rules	40
11.2 Examples	41
12 Security Considerations	41
13 Editors' Addresses	41
14 References	42
15 Full Copyright Statement	43

1. Introduction

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 an adapted subset of OSI's Abstract Syntax Notation One, ASN.1 (1988) [1]. It is the purpose of this document, the Structure of Management Information (SMI), to define that adapted subset, and to assign a set of associated administrative values.

The SMI is divided into three parts: module definitions, object definitions, and, notification 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 Structure of Management Information, as described in RFCs 1155 (STD 16), 1212 (STD 16), and RFC 1215, is termed the SMI version 1 (SMIv1). The current version of the Structure of Management Information is termed SMI version 2 (SMIv2).

2. Definitions

SNMPv2-SMI DEFINITIONS ::= BEGIN

-- the path to the root

```
org          OBJECT IDENTIFIER ::= { iso 3 }  -- "iso" = 1
dod          OBJECT IDENTIFIER ::= { org 6 }
internet     OBJECT IDENTIFIER ::= { dod 1 }
```

```
directory    OBJECT IDENTIFIER ::= { internet 1 }
```

```
mgmt         OBJECT IDENTIFIER ::= { internet 2 }
mib-2        OBJECT IDENTIFIER ::= { mgmt 1 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
```

```
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 }
```

-- Extended UTCTime, to allow dates with four-digit years

-- (Note that this definition of ExtUTCTime is not to be IMPORTed
-- by MIB modules.)

```
ExtUTCTime ::= OCTET STRING(SIZE(11 | 13))
```

-- format is YYMMDDHHMMZ or YYYYMMDDHHMMZ

```
--      where: YY      - last two digits of year (only years
--                    between 1900-1999)
--      YYYY - last four digits of the year (any year)
--      MM   - month (01 through 12)
--      DD   - day of month (01 through 31)
--      HH   - hours (00 through 23)
--      MM   - minutes (00 through 59)
--      Z    - denotes GMT (the ASCII character Z)
--
-- For example, "9502192015Z" and "199502192015Z" represent
-- 8:15pm GMT on 19 February 1995. Years after 1999 must use
-- the four digit year format. Years 1900-1999 may use the
-- two or four digit format.

-- definitions for information modules

MODULE-IDENTITY MACRO ::=
BEGIN
    TYPE NOTATION ::=
        "LAST-UPDATED" value(Update ExtUTCTime)
        "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 ExtUTCTime)
        "DESCRIPTION" Text

    -- a character string as defined in section 3.1.1
    Text ::= value(IA5String)
END

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

```

                                ReferPart

VALUE NOTATION ::=
                                value(VALUE OBJECT IDENTIFIER)

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

ReferPart ::=
                                "REFERENCE" Text
                                | empty

-- a character string as defined in section 3.1.1
Text ::= value(IA5String)
END

-- names of objects
-- (Note that these definitions of ObjectName and NotificationName
-- are not to be IMPORTed by MIB modules.)

ObjectName ::=
    OBJECT IDENTIFIER

NotificationName ::=
    OBJECT IDENTIFIER

-- syntax of objects

-- the "base types" defined here are:
--   3 built-in ASN.1 types: INTEGER, OCTET STRING, OBJECT IDENTIFIER
--   8 application-defined types: Integer32, IpAddress, Counter32,
--                               Gauge32, Unsigned32, TimeTicks, Opaque, and Counter64

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          -- includes Integer32
      INTEGER (-2147483648..2147483647),

    -- OCTET STRINGS with a more restrictive size
    -- may also be used
    string-value
      OCTET STRING (SIZE (0..65535)),

    objectID-value
      OBJECT IDENTIFIER
  }

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

-- application-wide types

ApplicationSyntax ::=
  CHOICE {
    ipAddress-value
      IPAddress,

    counter-value
      Counter32,

    timeticks-value
      TimeTicks,

    arbitrary-value
      Opaque,

    big-counter-value
      Counter64,

    unsigned-integer-value -- includes Gauge32
      Unsigned32
  }

-- 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)

-- an unsigned 32-bit quantity
-- indistinguishable from Gauge32
Unsigned32 ::=
    [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 counters that wrap in less than one hour with only 32 bits
Counter64 ::=
    [APPLICATION 6]
        IMPLICIT INTEGER (0..18446744073709551615)

-- definition for objects
OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::=
        "SYNTAX" Syntax
        UnitsPart
        "MAX-ACCESS" Access
        "STATUS" Status
        "DESCRIPTION" Text
        ReferPart
```


IndexPart
DefValPart

VALUE NOTATION ::=
value(VALUE ObjectName)

Syntax ::= -- Must be one of the following:
-- a base type (or its refinement),
-- a textual convention (or its refinement), or
-- a BITS pseudo-type
type
| "BITS" "{" NamedBits "}"

NamedBits ::= NamedBit
| NamedBits "," NamedBit

NamedBit ::= identifier "(" number ")" -- number is nonnegative

UnitsPart ::=
"UNITS" Text
| empty

Access ::=
"not-accessible"
| "accessible-for-notify"
| "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(ObjectName)
Entry ::=
    -- use the INDEX value of the
    -- correspondent OBJECT-TYPE invocation
    value(ObjectName)

DefValPart ::= "DEFVAL" "{" Defvalue "}"
    | empty

Defvalue ::= -- must be valid for the type specified in
    -- SYNTAX clause of same OBJECT-TYPE macro
    value(ObjectSyntax)
    | "{" BitsValue "}"

BitsValue ::= BitNames
    | empty

BitNames ::= BitName
    | BitNames "," BitName

BitName ::= identifier

-- a character string as defined in section 3.1.1
Text ::= value(IA5String)
END

-- definitions for notifications

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

    VALUE NOTATION ::=
        value(VALUE NotificationName)

    ObjectsPart ::=
        "OBJECTS" "{" Objects "}"
        | empty
    Objects ::=
        Object
```

```

Object ::=
    | Objects "," Object
    value(ObjectName)

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

ReferPart ::=
    "REFERENCE" Text
    | empty

-- a character string as defined in section 3.1.1
Text ::= value(IA5String)
END

-- definitions of administrative identifiers

zeroDotZero    OBJECT-IDENTITY
STATUS          current
DESCRIPTION
    "A value used for null identifiers."
::= { 0 0 }

```

END

3. Information Modules

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

The SMI describes how to use an adapted subset of ASN.1 (1988) 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 will normally 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.

The constructs of ASN.1 allowed in SMIV2 information modules include: the IMPORTS clause, value definitions for OBJECT IDENTIFIERS, type definitions for SEQUENCES (with restrictions), ASN.1 type assignments of the restricted ASN.1 types allowed in SMIV2, and instances of ASN.1 macros defined in this document and its companion documents [2, 3]. Additional ASN.1 macros must not be defined in SMIV2 information modules. SMIV1 macros must not be used in SMIV2 information modules.

The names of all standard information modules must be unique (but different versions of the same information module should have the same name). Developers of enterprise information modules are encouraged to choose names for their information modules that will have a low probability of colliding with standard or other enterprise information modules. An information module may not use the ASN.1 construct of placing an object identifier value between the module name and the "DEFINITIONS" keyword. For the purposes of this specification, an ASN.1 module name begins with an upper-case letter and continues with zero or more letters, digits, or hyphens, except that a hyphen can not be the last character, nor can there be two consecutive hyphens.

All information modules start with exactly one invocation of the MODULE-IDENTITY macro, which provides contact information as well as revision history to distinguish between versions of the same information module. This invocation must appear immediately after any IMPORTS 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. (Note that this definition of a descriptor applies to all macros defined in this memo and in [2].)

For the purposes of this specification, an ASN.1 identifier consists of one or more letters or digits, and its initial character must be a lower-case letter. Note that hyphens are not allowed by this specification (except for use by information modules converted from SMIV1 which did allow hyphens).

For all descriptors appearing in an information module, the descriptor shall be unique and mnemonic, and shall not exceed 64 characters in length. (However, descriptors longer than 32 characters are not recommended.) 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.

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 Values and Strings

Some clauses in a macro invocation may take a character string as a textual value (e.g., the DESCRIPTION clause). Other clauses take binary or hexadecimal strings (in any position where a non-negative number is allowed).

A character string is preceded and followed by the quote character ("), and consists of an arbitrary number (possibly zero) of:

- any 7-bit displayable ASCII characters except quote ("),
- tab characters,
- spaces, and
- line terminator characters (\n or \r\n).

The value of a character string is interpreted as ASCII.

A binary string consists of a number (possibly zero) of zeros and ones preceded by a single (') and followed by either the pair ('B) or ('b), where the number is a multiple of eight.

A hexadecimal string consists of an even number (possibly zero) of hexadecimal digits, preceded by a single (') and followed by either the pair ('H) or ('h). Digits specified via letters can be in upper or lower case.

Note that ASN.1 comments can not be enclosed inside any of these types of strings.

3.2. IMPORTing Symbols

To reference an external object, the IMPORTS statement must be used to identify both the descriptor and the module in which the descriptor is defined, where the module is identified by its ASN.1 module name.

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 to refer to objects even when there is no collision when IMPORTing symbols.

Finally, if any of the ASN.1 named types and macros defined in this document, specifically:

Counter32, Counter64, Gauge32, Integer32, IPAddress, MODULE-IDENTITY, NOTIFICATION-TYPE, Opaque, OBJECT-TYPE, OBJECT-IDENTITY, TimeTicks, Unsigned32,

or any of those defined in [2] or [3], are used in an information module, then they must be imported using the IMPORTS statement. However, the following must not be included in an IMPORTS statement:

- named types defined by ASN.1 itself, specifically: INTEGER, OCTET STRING, OBJECT IDENTIFIER, SEQUENCE, SEQUENCE OF type,
- the BITS construct.

3.3. Exporting Symbols

The ASN.1 EXPORTS statement is not allowed in SMIv2 information modules. All items defined in an information module are automatically exported.

3.4. ASN.1 Comments

ASN.1 comments can be included in an information module. However, it is recommended that all substantive descriptions be placed within an appropriate DESCRIPTION clause.

ASN.1 comments commence with a pair of adjacent hyphens and end with the next pair of adjacent hyphens or at the end of the line, whichever occurs first. Comments ended by a pair of hyphens have the effect of a single space character.

3.5. OBJECT IDENTIFIER values

An OBJECT IDENTIFIER value is an ordered list of non-negative numbers. For the SMIv2, each number in the list is referred to as a sub-identifier, there are at most 128 sub-identifiers in a value, and each sub-identifier has a maximum value of $2^{32}-1$ (4294967295 decimal).

All OBJECT IDENTIFIER values have at least two sub-identifiers, where the value of the first sub-identifier is one of the following well-known names:

Value	Name
0	ccitt
1	iso
2	joint-iso-ccitt

(Note that this SMI does not recognize "new" well-known names, e.g., as defined when the CCITT became the ITU.)

3.6. OBJECT IDENTIFIER usage

OBJECT IDENTIFIERS are used in information modules in two ways:

- (1) registration: the definition of a particular item is registered as a particular OBJECT IDENTIFIER value, and associated with a particular descriptor. After such a registration, the semantics thereby associated with the value are not allowed to change, the OBJECT IDENTIFIER can not be used for any other registration, and the descriptor can not be changed nor associated with any other registration. The following macros result in a registration:

OBJECT-TYPE, MODULE-IDENTITY, NOTIFICATION-TYPE, OBJECT-GROUP,
OBJECT-IDENTITY, NOTIFICATION-GROUP, MODULE-COMPLIANCE,
AGENT-CAPABILITIES.

- (2) assignment: a descriptor can be assigned to a particular OBJECT IDENTIFIER value. For this usage, the semantics associated with the OBJECT IDENTIFIER value is not allowed to change, and a descriptor assigned to a particular OBJECT IDENTIFIER value cannot subsequently be assigned to another. However, multiple descriptors can be assigned to the same OBJECT IDENTIFIER value. Such assignments are specified in the following manner:

```
mib          OBJECT IDENTIFIER ::= { mgmt 1 } -- from RFC1156
mib-2        OBJECT IDENTIFIER ::= { mgmt 1 } -- from RFC1213
fredRouter   OBJECT IDENTIFIER ::= { flintStones 1 1 }
barneySwitch OBJECT IDENTIFIER ::= { flintStones bedrock(2) 1 }
```

Note while the above examples are legal, the following is not:

```
dinoHost OBJECT IDENTIFIER ::= { flintStones bedrock 2 }
```

A descriptor is allowed to be associated with both a registration and an assignment, providing both are associated with the same OBJECT IDENTIFIER value and semantics.

3.7. Reserved Keywords

The following are reserved keywords which must not be used as descriptors or module names:

```
ABSENT ACCESS AGENT-CAPABILITIES ANY APPLICATION AUGMENTS BEGIN
BIT BITS BOOLEAN BY CHOICE COMPONENT COMPONENTS CONTACT-INFO
CREATION-REQUIRES Counter32 Counter64 DEFAULT DEFINED
DEFINITIONS DEFVAL DESCRIPTION DISPLAY-HINT END ENUMERATED
ENTERPRISE EXPLICIT EXPORTS EXTERNAL FALSE FROM GROUP Gauge32
IDENTIFIER IMPLICIT IMPLIED IMPORTS INCLUDES INDEX INTEGER
Integer32 IpAddress LAST-UPDATED MANDATORY-GROUPS MAX MAX-ACCESS
MIN MIN-ACCESS MINUS-INFINITY MODULE MODULE-COMPLIANCE MODULE-
IDENTITY NOTIFICATION-GROUP NOTIFICATION-TYPE NOTIFICATIONS NULL
OBJECT OBJECT-GROUP OBJECT-IDENTITY OBJECT-TYPE OBJECTS OCTET OF
OPTIONAL ORGANIZATION Opaque PLUS-INFINITY PRESENT PRIVATE
PRODUCT-RELEASE REAL REFERENCE REVISION SEQUENCE SET SIZE STATUS
STRING SUPPORTS SYNTAX TAGS TEXTUAL-CONVENTION TRAP-TYPE TRUE
TimeTicks UNITS UNIVERSAL Unsigned32 VARIABLES VARIATION WITH
WRITE-SYNTAX
```

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.

Note that reference in an IMPORTS clause or in clauses of SMIv2 macros to an information module is NOT through the use of the 'descriptor' of a MODULE-IDENTITY macro; rather, an information module is referenced through specifying its module name.

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 (including the initial version) made to this information module, in reverse chronological order (i.e., most recent first). Each instance of this clause contains the date and time of the revision.

5.5.1. Mapping of the DESCRIPTION sub-clause

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

5.6. 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 specifying an OBJECT IDENTIFIER value to refer to the information module containing the invocation.

Note that it is a common practice to use the value of the MODULE-IDENTITY macro as a subtree under which other OBJECT IDENTIFIER values assigned within the module are defined. However, it is legal (and occasionally necessary) for the other OBJECT IDENTIFIER values assigned within the module to be unrelated to the OBJECT IDENTIFIER value of the MODULE-IDENTITY macro.

5.7. 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 "199505241811Z"

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 "9505241811Z"

DESCRIPTION

"The latest version of this MIB module."

REVISION "9210070433Z"

DESCRIPTION

"The initial version of this MIB module, published in
RFC yyyy."

-- 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. All administrative OBJECT IDENTIFIER assignments which define a type identification value (see AutonomousType, a textual convention defined in [3]) should be defined via the OBJECT-IDENTITY macro. 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 value "current" means that the definition is current and valid. The value "obsolete" means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value "deprecated" also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

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 some other document, either another information module which defines a related assignment, or some other document which provides additional information relevant to this definition.

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 type of 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.

For leaf objects which are not columnar objects (i.e., not contained within a conceptual table), 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.

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 following: a base type, the BITS construct, or a textual convention. (SEQUENCE OF and SEQUENCE are also possible for conceptual tables, see section 7.1.12). The base types are those defined in the ObjectSyntax CHOICE. A textual convention is a newly-defined type defined as a sub-type of a base type [3].

An extended subset of the full capabilities of ASN.1 (1988) sub-typing is allowed, as appropriate to the underlying ASN.1 type. Any such restriction on size, range or enumerations specified in this clause represents the maximal level of support which makes "protocol sense". Restrictions on sub-typing are specified in detail in Section 9 and Appendix A of this memo.

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. Both the INTEGER and Integer32 types may be sub-typed to be more constrained than the Integer32 type.

The INTEGER type (but not the Integer32 type) may also be used to represent integer-valued information as named-number enumerations. In this 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, a label for a named-number enumeration must consist of one or more letters or digits, up to a maximum of 64 characters, and the initial character must be a lower-case letter. (However, labels longer than 32 characters are not recommended.) Note that hyphens are not allowed by this specification (except for use by information modules converted from SMIv1 which did allow hyphens).

7.1.2. OCTET STRING

The OCTET STRING type represents arbitrary binary or textual data. Although the SMI-specified size limitation for this type is 65535 octets, MIB designers should realize that there may be implementation and interoperability limitations for sizes in excess of 255 octets.

7.1.1.3. 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).

7.1.1.4. The BITS construct

The BITS construct represents an enumeration of named bits. This collection is assigned non-negative, contiguous (but see below) values, starting at zero. Only those named-bits so enumerated may be present in a value. (Thus, enumerations must be assigned to consecutive bits; however, see Section 9 for refinements of an object with this syntax.)

As part of updating an information module, for an object defined using the BITS construct, new enumerations can be added or existing enumerations can have new labels assigned to them. After an enumeration is added, it might not be possible to distinguish between an implementation of the updated object for which the new enumeration is not asserted, and an implementation of the object prior to the addition. Depending on the circumstances, such an ambiguity could either be desirable or could be undesirable. The means to avoid such an ambiguity is dependent on the encoding of values on the wire; however, one possibility is to define new enumerations starting at the next multiple of eight bits. (Of course, this can also result in the enumerations no longer being contiguous.)

Although there is no SMI-specified limitation on the number of enumerations (and therefore on the length of a value), except as may be imposed by the limit on the length of an OCTET STRING, MIB designers should realize that there may be implementation and interoperability limitations for sizes in excess of 128 bits.

Finally, a label for a named-number enumeration must consist of one or more letters or digits, up to a maximum of 64 characters, and the initial character must be a lower-case letter. (However, labels longer than 32 characters are not recommended.) Note that hyphens are not allowed by this specification.

7.1.1.5. 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 an appropriate SYNTAX clause, to indicate the last discontinuity. Examples of appropriate SYNTAX clause include: `TimeStamp` (a textual convention defined in [3]), `DateAndTime` (another textual convention from [3]) or `TimeTicks`.

The value of the `MAX-ACCESS` clause for objects with a SYNTAX clause value of `Counter32` is either "read-only" or "accessible-for-notify".

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, nor fall below a minimum value. The maximum value can not be greater than $2^{32}-1$ (4294967295 decimal), and the minimum value can not be smaller than 0. The value of a `Gauge32` has its maximum value whenever the information being modeled is greater than or equal to its maximum value, and has its minimum value whenever the information being modeled is smaller than or equal to its minimum value. If the information being modeled subsequently decreases below (increases above) the maximum (minimum) value, the `Gauge32` also decreases (increases). (Note that despite of the use of the term "latched" in the original definition of this type, it does not become "stuck" at its maximum or minimum value.)

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 the time when sysUpTime [5] was zero, and the second reference epoch is defined as the current value of sysUpTime.

The TimeTicks type may not be sub-typed.

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. 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 an appropriate SYNTAX clause, to indicate the last discontinuity. Examples of appropriate SYNTAX

clause are: `TimeStamp` (a textual convention defined in [3]), `DateAndTime` (another textual convention from [3]) or `TimeTicks`.

The value of the `MAX-ACCESS` clause for objects with a `SYNTAX` clause value of `Counter64` is either "read-only" or "accessible-for-notify".

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`.

7.1.11. Unsigned32

The `Unsigned32` type represents integer-valued information between 0 and $2^{32}-1$ inclusive (0 to 4294967295 decimal).

7.1.12. Conceptual Tables

Management operations apply exclusively to scalar objects. However, it is sometimes convenient for developers of management applications to impose an imaginary, tabular structure on an ordered collection of objects within 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,

except that both sub-typing information and the named values for enumerated integers or the named bits for the BITS construct, are omitted from <syntax>.

Further, a <type> is always present for every subordinate object. (The ASN.1 DEFAULT and OPTIONAL clauses are disallowed in the SEQUENCE definition.) The MAX-ACCESS clause for conceptual tables and rows is "not-accessible".

7.1.12.1. Creation and Deletion of Conceptual Rows

For newly-defined conceptual rows which allow the creation of new object instances and/or 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.

7.2. 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, or to include its value in a notification. 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 an auxiliary object (see Section 7.7). The value "accessible-for-notify" indicates an object which is accessible only via a notification (e.g., snmpTrapOID [5]).

These values are ordered, from least to greatest: "not-accessible", "accessible-for-notify", "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 value "current" means that the definition is current and valid. The value "obsolete" means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value "deprecated" also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing 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 some other document, either another information module which defines a related assignment, or some other document which provides additional information relevant to this definition.

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.

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 objects can be columnar objects from the same and/or another conceptual table, but must not be scalar objects. Multiple occurrences of the same object in a single INDEX clause is strongly discouraged.

The syntax of the objects in the INDEX clause indicate how to form the instance-identifier:

- (1) integer-valued (i.e., having INTEGER as its underlying primitive type): 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 (when preceded by the IMPLIED keyword): 'n' sub-identifiers, where 'n' is the number of sub-identifiers in the value (each sub-identifier of the value is copied into a separate sub-identifier);
- (5) object identifier-valued (when not preceded by the IMPLIED keyword): '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);
- (6) IPAddress-valued: 4 sub-identifiers, in the familiar a.b.c.d notation.

Note that the IMPLIED keyword can only be present for an object having a variable-length syntax (e.g., variable-length strings or object identifier-valued objects). Further, the IMPLIED keyword can only be associated with the last object in the INDEX clause. Finally, the IMPLIED keyword may not be used on a variable-length string object if that string might have a value of zero-length.

Since a single value of a Counter has (in general) no information content (see section 7.1.6 and 7.1.10), objects defined using the syntax, Counter32 or Counter64, must not be specified in an INDEX

clause. If an object defined using the BITS construct is used in an INDEX clause, it is considered a variable-length string.

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 auxiliary objects is "not-accessible", except in the following circumstances:

- (1) within a MIB module originally written to conform to SMIv1, and later converted to conform to SMIv2; or
- (2) a conceptual row must contain at least one columnar object which is not an auxiliary object. In the event that all of a conceptual row's columnar objects are also specified in its INDEX clause, then one of them must be accessible, i.e., have a MAX-ACCESS clause of "read-only". (Note that this situation does not arise for a conceptual row allowing create access, since such a row will have a status column which will not be an auxiliary object.)

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.

7.8. 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 augmentation. (Thus, a conceptual row augmentation cannot itself be augmented.) Instances of subordinate columnar objects of a conceptual row augmentation 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 augmentation 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. On the other hand, it would be incorrect to use the AUGMENTS clause for the relationship between RFC 2233's ifTable

and the many media-specific MIBs which extend it for specific media (e.g., the dot3Table in RFC 2358), since not all interfaces are of the same media.

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

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 conceptual rows of this table and an existing table, then an INDEX clause should be used which is identical to that in the existing table. For example, the relationship between RFC 2233's ifTable and a media-specific MIB which extends the ifTable for a specific media (e.g., the dot3Table in RFC 2358), is a sparse relationship.
- (3) Otherwise, if no existing objects have the required syntax and semantics, then 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 an agent when an object instance is created. That is, the value is a "hint" to implementors.

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 an agent might use (especially for a read-only object).

Note that with this definition of the DEFVAL clause, it is appropriate to use it for any columnar object of a read-create table. It is also permitted to use it for scalar objects dynamically created by an agent, or for columnar objects of a read-write table dynamically created by an agent.

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 an agent might use.

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

ObjectSyntax	DEFVAL clause
-----	-----
Integer32	DEFVAL { 1 } -- same for Gauge32, TimeTicks, Unsigned32
INTEGER	DEFVAL { valid } -- enumerated value
OCTET STRING	DEFVAL { 'ffffffffffff'H }
DisplayString	DEFVAL { "SNMP agent" }
IpAddress	DEFVAL { 'c0210415'H } -- 192.33.4.21
OBJECT IDENTIFIER	DEFVAL { sysDescr }
BITS	DEFVAL { { primary, secondary } }
	-- enumerated values that are set
BITS	DEFVAL { { } }
	-- no enumerated values are set

A binary string used in a DEFVAL clause for an OCTET STRING must be either an integral multiple of eight or zero bits in length; similarly, a hexadecimal string must be an even number of hexadecimal digits. The value of a character string used in a DEFVAL clause must not contain tab characters or line terminator characters.

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.

7.11. Usage Example

Consider how one might define a conceptual table and its subordinates. (This example uses the RowStatus textual convention defined in [3].)

evalSlot OBJECT-TYPE

SYNTAX Integer32 (0..2147483647)

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The index number of the first unassigned entry in the evaluation table, or the value of zero indicating that all entries are assigned.

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 createAndGo(4) or createAndWait(5). 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 (1..2147483647)
    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 evaluated, or zero if
         no such value is available."
    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 an ordered sequence of MIB object types. One and only one object instance for each occurrence of each object type must be present, and in the specified order, in every instance of the notification. If the same object type occurs multiple times in a notification's ordered sequence, then an object instance is present for each of them. An object type specified in this clause must not have an MAX-ACCESS clause of "not-accessible". The notification's DESCRIPTION clause must specify the information/meaning conveyed by each occurrence of each object type in the sequence. The DESCRIPTION clause must also specify which object instance is present for each object type in the notification.

Note that an agent is allowed, at its own discretion, to append as many additional objects as it considers useful to the end of the notification (i.e., after the objects defined by the OBJECTS clause).

8.2. Mapping of the STATUS clause

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

The value "current" means that the definition is current and valid. The value "obsolete" means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value "deprecated" also indicates an obsolete definition, it permits new/continued implementation in order to foster

interoperability with older/existing 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 notification. 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 some other document, either another information module which defines a related assignment, or some other document which provides additional information relevant to this definition.

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. In order to achieve compatibility with SNMPv1 traps, both when converting SMIv1 information modules to/from this SMI, and in the procedures employed by multi-lingual systems and proxy forwarding applications, the next to last sub-identifier in the name of any newly-defined notification must have the value zero.

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 configuration change notification might be described:

```
entityMIBTraps      OBJECT IDENTIFIER ::= { entityMIB 2 }
entityMIBTrapPrefix OBJECT IDENTIFIER ::= { entityMIBTraps 0 }
```

```
entConfigChange NOTIFICATION-TYPE
    STATUS          current
    DESCRIPTION
```

```
        "An entConfigChange trap is sent when the value of
        entLastChangeTime changes. It can be utilized by an NMS to
        trigger logical/physical entity table maintenance polls."
```

An agent must not generate more than one entConfigChange 'trap-event' in a five second period, where a 'trap-event' is the transmission of a single trap PDU to a list of trap destinations. If additional configuration changes occur within the five second 'throttling' period, then these trap-events should be suppressed by the agent. An NMS should periodically check the value of entLastChangeTime to detect any missed entConfigChange trap-events, e.g. due to throttling or transmission loss."

```
::= { entityMIBTrapPrefix 1 }
```

According to this invocation, the notification authoritatively identified as

```
{ entityMIBTrapPrefix 1 }
```

is used to report a particular type of configuration change.

9. Refined Syntax

Some macros have clauses which allows syntax to be refined, specifically: the SYNTAX clause of the OBJECT-TYPE macro, and the SYNTAX/WRITE-SYNTAX clauses of the MODULE-COMPLIANCE and AGENT-CAPABILITIES macros [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:

object syntax	Restrictions to Refinement of		
	range	enumeration	size
-----	-----	-----	----
INTEGER	(1)	(2)	-
Integer32	(1)	-	-
Unsigned32	(1)	-	-
OCTET STRING	-	-	(3)
OBJECT IDENTIFIER	-	-	-
BITS	-	(2)	-
IpAddress	-	-	-
Counter32	-	-	-
Counter64	-	-	-
Gauge32	(1)	-	-
TimeTicks	-	-	-

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 (note that for BITS, a refinement may cause the enumerations to no longer be contiguous); or,
- (3) the size in octets of the value may be refined by raising the lower-bounds, by reducing the upper-bounds, and/or by reducing the alternative size choices.

No other types of refinements can be specified in the SYNTAX clause. However, the DESCRIPTION clause is available to specify additional restrictions which can not be expressed in the SYNTAX clause. Further details on (and examples of) sub-typing are provided in Appendix A.

10. Extending an Information Module

As experience is gained with an information module, it may be desirable to revise that information module. However, changes are not allowed if they have any potential to cause interoperability problems "over the wire" between an implementation using an original specification and an implementation using an updated specification(s).

For any change, the invocation of the MODULE-IDENTITY macro must be updated to include information about the revision: specifically, updating the LAST-UPDATED clause, adding a pair of REVISION and DESCRIPTION clauses (see section 5.5), and making any necessary changes to existing clauses, including the ORGANIZATION and CONTACT-INFO clauses.

Note that any definition contained in an information module is available to be IMPORT-ed by any other information module, and is referenced in an IMPORTS clause via the module name. Thus, a module name should not be changed. Specifically, the module name (e.g., "FIZBIN-MIB" in the example of Section 5.7) should not be changed when revising an information module (except to correct typographical errors), and definitions should not be moved from one information module to another.

Also note that obsolete definitions must not be removed from MIB modules since their descriptors may still be referenced by other information modules, and the OBJECT IDENTIFIERS used to name them must never be re-assigned.

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. Similarly, named bits may be added or existing labels changed for the BITS construct.
- (2) The value of a SYNTAX clause may be replaced by a textual convention, providing the textual convention is defined to use the same primitive ASN.1 type, has the same set of values, and has identical semantics.
- (3) 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". When making such a change, the DESCRIPTION clause should be updated to explain the rationale.
- (4) A DEFVAL clause may be added or updated.
- (5) A REFERENCE clause may be added or updated.
- (6) A UNITS clause may be added.
- (7) A conceptual row may be augmented by adding new columnar objects at the end of the row, and making the corresponding update to the SEQUENCE definition.
- (8) Clarifications and additional information may be included in the DESCRIPTION clause.
- (9) 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 than 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.

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.
- (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". When making such a change, the DESCRIPTION clause should be updated to explain the rationale.
- (3) A DESCRIPTION clause may be clarified.

Otherwise, if the semantics of any previously defined notification are changed (i.e., if a non-editorial change is made to any clause other than 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.

11. Appendix A: Detailed Sub-typing Rules

11.1. Syntax Rules

The syntax rules for sub-typing are given below. Note that while this syntax is based on ASN.1, it includes some extensions beyond what is allowed in ASN.1, and a number of ASN.1 constructs are not allowed by this syntax.

```
<integerSubType>
  ::= <empty>
     | "(" <range> ["|" <range>]... ")"

<octetStringSubType>
  ::= <empty>
     | "(" "SIZE" "(" <range> ["|" <range>]... ")" ")"

<range>
  ::= <value>
     | <value> ".." <value>

<value>
  ::= "-" <number>
     | <number>
     | <hexString>
     | <binString>
```

where:

```
<empty>      is the empty string
<number>     is a non-negative integer
<hexString>  is a hexadecimal string (e.g., '0F0F'H)
<binString>  is a binary string (e.g., '1010'B)
```

<range> is further restricted as follows:

- any <value> used in a SIZE clause must be non-negative.
- when a pair of values is specified, the first value must be less than the second value.
- when multiple ranges are specified, the ranges may not overlap but may touch. For example, (1..4 | 4..9) is invalid, and (1..4 | 5..9) is valid.
- the ranges must be a subset of the maximum range of the base type.

11.2. Examples

Some examples of legal sub-typing:

```
Integer32 (-20..100)
Integer32 (0..100 | 300..500)
Integer32 (300..500 | 0..100)
Integer32 (0 | 2 | 4 | 6 | 8 | 10)
OCTET STRING (SIZE(0..100))
OCTET STRING (SIZE(0..100 | 300..500))
OCTET STRING (SIZE(0 | 2 | 4 | 6 | 8 | 10))
SYNTAX TimeInterval (0..100)
SYNTAX DisplayString (SIZE(0..32))
```

(Note the last two examples above are not valid in a TEXTUAL CONVENTION, see [3].)

Some examples of illegal sub-typing:

```
Integer32 (150..100)           -- first greater than second
Integer32 (0..100 | 50..500)  -- ranges overlap
Integer32 (0 | 2 | 0 )        -- value duplicated
Integer32 (MIN..-1 | 1..MAX)  -- MIN and MAX not allowed
Integer32 (SIZE (0..34))      -- must not use SIZE
OCTET STRING (0..100)         -- must use SIZE
OCTET STRING (SIZE(-10..100)) -- negative SIZE
```

12. Security Considerations

This document defines a language with which to write and read descriptions of management information. The language itself has no security impact on the Internet.

13. Editors' Addresses

Keith McCloghrie
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
Phone: +1 408 526 5260
EMail: kzm@cisco.com

David Perkins
SNMPinfo
3763 Benton Street
Santa Clara, CA 95051
USA
Phone: +1 408 221-8702
EMail: dperkins@snmpinfo.com

Juergen Schoenwaelder
TU Braunschweig
Bueltenweg 74/75
38106 Braunschweig
Germany
Phone: +49 531 391-3283
EMail: schoenw@ibr.cs.tu-bs.de

14. 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., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.
- [3] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.
- [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] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907, January 1996.
- [6] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

15. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."