            Next Generation Structure of Management Information (SMIng)
             Mappings to the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

   SMIng (Structure of Management Information, Next Generation)
   (RFC3780), is a protocol-independent data definition language for
   management information. This memo defines an SMIng language
   extension that specifies the mapping of SMIng definitions of
   identities, classes, and their attributes and events to dedicated
   definitions of nodes, scalar objects, tables and columnar objects,
   and notifications, for application to the SNMP management framework.

Table of Contents

1. Introduction

   SMIng (Structure of Management Information, Next Generation)
   [RFC3780] is a protocol-independent data definition language for
   management information. This memo defines an SMIng language
   extension that specifies the mapping of SMIng definitions of
   identities, classes, and their attributes and events to dedicated
   definitions of nodes, scalar objects, tables and columnar objects,
   and notifications for application in the SNMP management framework.
   Section 2 introduces basics of the SNMP management framework.
   Section 3 defines how SMIng data types are mapped to the data types
   supported by the SNMP protocol. It introduces some new ASN.1 [ASN1]
   definitions which are used to represent new SMIng base types such as
   floats in the SNMP protocol.

   Section 4 describes the semantics of the SNMP mapping extensions for
   SMIng. The formal SMIng specification of the extension is provided
   in Section 5.

   Section 6 contains an SMIng module which defines derived types (such
   as RowStatus) that are specific to the SNMP mapping.

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119].

2. SNMP-Based Internet Management

   The SNMP network management framework [RFC3410] is based on the
   concept of "managed objects". Managed objects represent real or
   synthesized variables of systems that are to be managed. Note that
   in spite of these terms this model is not object-oriented. For
   naming purposes, the managed objects are organized hierarchically in
   an "object identifier tree", where only leaf nodes may represent
   objects.

   Nodes in the object identifier tree may also identify conceptual
   tables, rows of conceptual tables, notifications, groups of objects
   and/or notifications, compliance statements, modules or other
   information. Each node is identified by an unique "object
   identifier" value which is a sequence of non-negative numbers, named
   "sub-identifiers", where the left-most sub-identifier refers to the

node next to the root of the tree and the right-most sub-identifier
refers to the node that is identified by the complete object
identifier value. Each sub-identifier has a value between 0 and
2^32-1 (4294967295).

The SMIng extensions described in this document are used to map SMIng
data definitions to SNMP compliant managed objects. This mapping is
designed to be readable to computer programs, named MIB compilers, as
well as to human readers.

2.1. Kinds of Nodes

Each node in the object identifier tree is of a certain kind and may
represent management information or not:

o Simple nodes, that do not represent management information, but
   may be used for grouping nodes in a subtree. Those nodes are
   defined by the `node' statement. This statement can also be used
   to map an SMIng `identity' to a node.

o Nodes representing the identity of a module to allow references to
   a module in other objects of type `ObjectIdentifier'. Those nodes
   are defined by the `snmp' statement,

o Scalar objects, which have exactly one object instance and no
   child nodes. See Section 2.2 for scalar objects' instances. A
   set of scalar objects is mapped from one or more SMIng classes
   using the `scalars' statement. The statement block of the
   `scalars' statement contains one `implements' statement for each
   class. The associated statement blocks in turn contain `object'
   statements that specify the mapping of attributes to scalar
   objects. Scalar objects MUST not have any child node.

o Tables, which represent the root node of a collection of
   information structured in table rows. Table nodes are defined by
   the `table' statement. A table object identifier SHOULD not have
   any other child node than the implicitly defined row node (see
   below).

o Rows, which belong to a table (that is, row's object identifier
   consists of the table's full object identifier plus a single `1'
   sub-identifier) and represent a sequence of one or more columnar
   objects. A row node is implicitly defined for each table node.

o Columnar objects, which belong to a row (that is, the columnar
   objects' object identifier consists of the row's full object
   identifier plus a single column-identifying sub-identifier) and
   have zero or more object instances and no child nodes. They are
   defined as follows: The classes that are implemented by a `table'
   statement are identified by `implements' statements. The
   statement block of each `implements' statement contains `object'
   statements that specify the mapping of attributes to columnar
   objects of this table. Columnar objects MUST not have any child
   node.

o Notifications, which represent information that is sent by agents
   within unsolicited transmissions. The `notification' statement is
   used to map an SMIng event to a notification. A notification's
   object identifier SHOULD not have any child node.

o Groups of objects and notifications, which may be used for
   compliance statements. They are defined using the `group'
   statement.

o Compliance statements which define requirements for MIB module
   implementations. They are defined using the `compliance'
   statement.

2.2. Scalar and Columnar Object Instances

   Instances of managed objects are identified by appending an
   instance-identifier to the object's object identifier. Scalar
   objects and columnar objects use different ways to construct the
   instance-identifier.

   Scalar objects have exactly one object instance. It is identified by
   appending a single `0' sub-identifier to the object identifier of the
   scalar object.

   Within tables, different instances of the same columnar object are
   identified by appending a sequence of one or more sub-identifiers to
   the object identifier of the columnar object which consists of the
   values of object instances that unambiguously distinguish a table
   row. These indexing objects can be columnar objects of the same
   and/or another table, but MUST NOT be scalar objects. Multiple
   applications of the same object in a single table indexing
   specification are strongly discouraged.

The base types of the indexing objects indicate how to form the
instance-identifier:

o integer-valued or enumeration-valued: a single sub-identifier
   taking the integer value (this works only for non-negative
   integers and integers of a size of up to 32 bits),

o string-valued, fixed-length strings (or variable-length with
   compact encoding): `n' sub-identifiers, where `n' is the length of
   the string (each octet of the string is encoded in a separate
   sub-identifier),

o string-valued, variable-length strings or bits-valued: `n+1' sub-
   identifiers, where `n' is the length of the string or bits
   encoding (the first sub-identifier is `n' itself, following this,
   each octet of the string or bits is encoded in a separate sub-
   identifier),

o object identifier-valued (with compact encoding): `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),

o 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),

Note that compact encoding can only be applied to an object having a
variable-length syntax (e.g., variable-length strings, bits objects
or object identifier-valued objects). Further, compact encoding can
only be associated with the last object in a list of indexing
objects. Finally, compact encoding MUST 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 or enumeration-valued
objects are RECOMMENDED to be numbered starting from one (i.e., not
from zero). Integer objects that allow negative values, Unsigned64
objects, Integer64 objects and floating point objects MUST NOT be
used for table indexing.

Objects which are both specified for indexing in a row and also
columnar objects of the same row are termed auxiliary objects.
Auxiliary objects SHOULD be non-accessible, except in the following
circumstances:

o within a module originally written to conform to SMIv1, or

o a row must contain at least one columnar object which is not an
   auxiliary object. In the event that all of a row's columnar
   objects are also specified to be indexing objects then one of them
   MUST be accessible.

2.3. Object Identifier Hierarchy

The layers of the object identifier tree near the root are well
defined and organized by standardization bodies. The first level
next to the root has three nodes:

   0: ccitt

   1: iso

   2: joint-iso-ccitt

Note that the renaming of the Commite Consultatif International de
Telegraphique et Telephonique (CCITT) to International
Telecommunications Union (ITU) had no consequence on the names used
in the object identifier tree.

The root of the subtree administered by the Internet Assigned Numbers
Authority (IANA) for the Internet is `1.3.6.1' which is assigned with
the identifier `internet'. 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:

The `mgmt' (internet.2) subtree is used to identify "standard"
definitions. An information module produced by an IETF working group
becomes a "standard" information module when the document is first
approved by the IESG and enters the Internet standards track.

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

The `private' (internet.4) subtree is used to identify definitions
defined unilaterally. The `enterprises' (private.1) subtree beneath
private is used, among other things, to permit providers of
networking subsystems to register information modules of their
products.

These and some other nodes are defined in the SMIng module NMRG-
SMING-SNMP-EXT (Section 5).

3. SMIng Data Type Mappings

SMIng [RFC3780] supports the following set of base types:
OctetString, Pointer, Integer32, Integer64, Unsigned32, Unsigned64,
Float32, Float64, Float128, Enumeration, Bits, and ObjectIdentifier.

The SMIng core module NMRG-SMING ([RFC3780], Appendix A) defines
additional derived types, among them Counter32 (derived from
Unsigned32), Counter64 (derived from Unsigned64), TimeTicks32 and
TimeTicks64 (derived from Unsigned32 and Unsigned64), IpAddress
(derived from OctetString), and Opaque (derived from OctetString).

The version 2 of the protocol operations for SNMP document [RFC3416]
defines the following 9 data types which are distinguished by the
protocol: INTEGER, OCTET STRING, OBJECT IDENTIFIER, IpAddress,
Counter32, TimeTicks, Opaque, Counter64, and Unsigned32.

The SMIng base types and their derived types are mapped to SNMP data
types according to the following table:

SMIng Data Type	SNMP Data Type	Comment
OctetString	OCTET STRING	(1)
Pointer	OBJECT IDENTIFIER	
Integer32	INTEGER	
Integer64	Opaque (Integer64)	(2)
Unsigned32	Unsigned32	(3)
Unsigned64	Opaque (Unsigned64)	(2) (4)
Float32	Opaque (Float32)	(2)
Float64	Opaque (Float64)	(2)
Float128	Opaque (Float128)	(2)
Enumeration	INTEGER	
Bits	OCTET STRING	
ObjectIdentifier	OBJECT IDENTIFIER	
Counter32	Counter32	
Counter64	Counter64	
TimeTicks32	TimeTicks	
TimeTicks64	Opaque (Unsigned64)	(2)
IpAddress	IpAddress	
Opaque	Opaque	

(1) This mapping includes all types derived from the OctetString
    type except those types derived from the IpAddress and Opaque
    SMIng types defined in the module NMRG-SMING.

        (2) This type is encoded according to the ASN.1 type with the same
            name defined in Section 3.1. The resulting BER encoded value
            is then wrapped in an Opaque value.

        (3) This mapping includes all types derived from the Unsigned32
            type except those types derived from the Counter32 and
            TimeTicks32 SMIng types defined in the module NMRG-SMING.

        (4) This mapping includes all types derived from the Unsigned64
            type except those types derived from the Counter64 SMIng type
            defined in the module NMRG-SMING.

3.1. ASN.1 Definitions

   The ASN.1 [ASN1] type definitions below introduce data types which
   are used to map the new SMIng base types into the set of ASN.1 types
   supported by the second version of SNMP protocol operations
   [RFC3416].

   NMRG-SMING-SNMP-MAPPING DEFINITIONS ::= BEGIN

   Integer64 ::=
       [APPLICATION 10]
           IMPLICIT INTEGER (-9223372036854775808..9223372036854775807)

   Unsigned64
       [APPLICATION 11]
           IMPLICIT INTEGER (0..18446744073709551615)

   Float32
       [APPLICATION 12]
           IMPLICIT OCTET STRING (SIZE (4))

   Float64
       [APPLICATION 13]
           IMPLICIT OCTET STRING (SIZE (8))

   Float128
       [APPLICATION 14]
           IMPLICIT OCTET STRING (SIZE (16))

   END

   The definitions of Integer64 and Unsigned64 are consistent with the
   same definitions in the SPPI [RFC3159]. The floating point types
   Float32, Float64 and Float128 support single, double and quadruple

IEEE floating point values. The encoding of the values follows the
"IEEE Standard for Binary Floating-Point Arithmetic" as defined in
ANSI/IEEE Standard 754-1985 [IEEE754].

4. The snmp Extension Statement

The `snmp' statement is the main statement of the SNMP mapping
specification. It gets one or two arguments: an optional lower-case
identifier that specifies a node that represents the module's
identity, and a mandatory statement block that contains all details
of the SNMP mapping. All information of an SNMP mapping are mapped
to an SNMP conformant module of the same name as the containing SMIng
module. A single SMIng module must not contain more than one `snmp'
statement.

4.1. The oid Statement

The snmp's `oid' statement, which must be present, if the snmp
statement contains a module identifier and must be absent otherwise,
gets one argument which specifies the object identifier value that is
assigned to this module's identity node.

4.2. The node Statement

The `node' statement is used to name and describe a node in the
object identifier tree, without associating any class or attribute
information with this node. This may be useful to group definitions
in a subtree of related management information, or to uniquely define
an SMIng `identity' to be referenced in attributes of type Pointer.
The `node' statement gets two arguments: a lower-case node identifier
and a statement block that holds detailed node information in an
obligatory order.

See the `nodeStatement' rule of the grammar (Section 5) for the
formal syntax of the `node' statement.

4.2.1. The node's oid Statement

The node's `oid' statement, which must be present, gets one argument
which specifies the object identifier value that is assigned to this
node.

4.2.2. The node's represents Statement

The node's `represents' statement, which need not be present, makes
this node represent an SMIng identity, so that objects of type
Pointer can reference that identity. The statement gets one argument
which specifies the identity name.

4.2.3 The node's status Statement

The node's `status' statement, which must be present, gets one
argument which is used to specify whether this node 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.

4.2.4. The node's description Statement

The node's `description' statement, which need not be present, gets
one argument which is used to specify a high-level textual
description of this node.

It is RECOMMENDED to include all semantics and purposes of this node.

4.2.5. The node's reference Statement

The node's `reference' statement, which need not be present, gets one
argument which is used to specify a textual cross-reference to some
other document, either another module which defines related
definitions, or some other document which provides additional
information relevant to this node.

4.2.6. Usage Examples

```
node iso                                 { oid 1;     status current; };
node   org                               { oid iso.3; status current; };
node     dod                             { oid org.6; status current; };
node       internet                      { oid dod.1; status current; };

node   zeroDotZero {
    oid        0.0;
    represents  NMRG-SMING::null;
    status      current;
    description "A null value used for pointers.";
};
```

4.3. The scalars Statement

The `scalars' statement is used to define the mapping of one or more
classes to a group of SNMP scalar managed objects organized under a
common parent node. The `scalars' statement gets two arguments: a

lower-case scalar group identifier and a statement block that holds
detailed mapping information of this scalar group in an obligatory
order.

See the `scalarsStatement' rule of the grammar (Section 5) for the
formal syntax of the `scalars' statement.

4.3.1. The scalars' oid Statement

The scalars' `oid' statement, which must be present, gets one
argument which specifies the object identifier value that is assigned
to the common parent node of this scalar group.

4.3.2. The scalars' object Statement

The scalars' `object' statement, which must be present at least once,
makes this scalar group contain a given scalar object. It gets two
arguments: the name of the scalar object to be defined and a
statement block that holds additional detailed information in an
obligatory order.

4.3.2.1. The object's implements Statement

The `implements' statement, which must be present, is used to specify
a single leaf attribute of a class that is implemented by this scalar
object. The type of this attribute must be a simple type, i.e., not
a class.

4.3.2.2. The object's subid Statement

The `subid' statement, which need not be present, is used to specify
the sub-identifier that identifies the scalar object within this
scalar group, i.e., the object identifier of the scalar object is the
concatenation of the values of this scalar group's oid statement and
of this subid statement.

If this statement is omitted, the sub-identifier is the one of the
previous object statement within this scalar group plus 1. If the
containing object statement is the first one within the containing
scalar group and the subid statement is omitted, the sub-identifier
is 1.

4.3.2.3. The object's status Statement

The object's `status' statement, which need not be present, gets one
argument which is used to specify whether this scalar object
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.

Scalar objects SHOULD NOT be defined as `current' if the implemented
attribute definition is `deprecated' or `obsolete'. Similarly, they
SHOULD NOT be defined as `deprecated' if the implemented attribute is
`obsolete'. Nevertheless, subsequent revisions of used class
definitions cannot be avoided, but SHOULD be taken into account in
subsequent revisions of the local module.

Note that it is RECOMMENDED to omit the status statement which means
that the status is inherited from the containing scalars statement.
However, if the status of a scalar object varies from the containing
scalar group, it has to be expressed explicitly, e.g., if the
implemented attribute has been deprecated or obsoleted.

4.3.2.4. The object's description Statement

The object's `description' statement, which need not be present, gets
one argument which is used to specify a high-level textual
description of this scalar object.

Note that in contrast to other definitions this description statement
is not mandatory and it is RECOMMENDED to omit it, if the object is
fully described by the description of the implemented attribute.

4.3.2.5. The object's reference Statement

The object's `reference' statement, which need not be present, gets
one argument which is used to specify a textual cross-reference to
some other document, either another module which defines related
definitions, or some other document which provides additional
information relevant to this scalar object.

It is RECOMMENDED to omit this statement, if the object's references
are fully described by the implemented attribute.

4.3.3. The scalars' status Statement

The scalars' `status' statement, which must be present, gets one
argument which is used to specify whether this scalar group
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.

4.3.4. The scalars' description Statement

   The scalars' `description' statement, which must be present, gets one
   argument which is used to specify a high-level textual description of
   this scalar group.

   It is RECOMMENDED to include all semantic definitions necessary for
   the implementation of this scalar group.

4.3.5. The scalars' reference Statement

   The scalars' `reference' statement, which need not be present, gets
   one argument which is used to specify a textual cross-reference to
   some other document, either another module which defines related
   definitions, or some other document which provides additional
   information relevant to this scalars statement.

4.3.6. Usage Example

```
   scalars ip {
     oid                mib-2.4;
     object ipForwarding { implements Ip.forwarding; };
     object ipDefaultTTL { implements Ip.defaultTTL; };
     // ...
     status             current;
     description
           "This scalar group implements the Ip class.";
   };
```

4.4. The table Statement

   The `table' statement is used to define the mapping of one or more
   classes to a single SNMP table of columnar managed objects. The
   `table' statement gets two arguments: a lower-case table identifier
   and a statement block that holds detailed mapping information of this
   table in an obligatory order.

   See the `tableStatement' rule of the grammar (Section 5) for the
   formal syntax of the `table' statement.

4.4.1. The table's oid Statement

The table's `oid' statement, which must be present, gets one argument
which specifies the object identifier value that is assigned to this
table's node.

4.4.2. Table Indexing Statements

SNMP table mappings offers five methods to supply table indexing
information: ordinary tables, table augmentations, sparse table
augmentations, table expansions, and reordered tables use different
statements to denote their indexing information. Each table
definition must contain exactly one of the following indexing
statements.

4.4.2.1. The table's index Statement for Table Indexing

The table's `index' statement, which is used to supply table indexing
information of base tables, gets one argument that specifies a
comma-separated list of objects, that are used for table indexing,
enclosed in parenthesis.

The elements of the `unique' statement of the implemented class(es)
and their order should be regarded as a hint for the index elements
of the table.

In case of modules that should be compatible on the SNMP protocol
level to SMIv2 versions of the module, an optional `implied' keyword
may be added in front of the list to indicate a compact encoding of
the last object in the list. See Section 2.2 for details.

4.4.2.2. The table's augments Statement for Table Indexing

The table's `augments' statement, which is used to supply table
indexing information of tables that augment a base table, gets one
argument that specifies the identifier of the table to be augmented.
Note that a table augmentation cannot itself be augmented. Anyhow, a
base table may be augmented by multiple table augmentations.

A table augmentation makes instances of subordinate columnar objects
identified according to the index specification of the base table
corresponding to the table named in the `augments' statement.
Further, instances of subordinate columnar objects of a table
augmentation exist according to the same semantics as instances of
subordinate columnar objects of the base table being augmented. As
such, note that creation of a base table row implies the

correspondent creation of any table row augmentations. Table
augmentations MUST NOT be used in table row creation and deletion
operations.

4.4.2.3. The table's extends Statement for Table Indexing

The table's `extends' statement, which is used to supply table
indexing information of tables that sparsely augment a base table,
gets one argument that specifies the identifier of the table to be
sparsely augmented. Note that a sparse table augmentation cannot
itself be augmented. Anyhow, a base table may be augmented by
multiple table augmentations, sparsely or not.

A sparse table augmentation makes instances of subordinate columnar
objects identified, if present, according to the index specification
of the base table corresponding to the table named in the `extends'
statement. Further, instances of subordinate columnar objects of a
sparse table augmentation exist according to the semantics as
instances of subordinate columnar objects of the base table and the
(non-formal) rules that confine the sparse relationship. As such,
note that creation of a sparse table row augmentation may be implied
by the creation of a base table row as well as done by an explicit
creation. However, if a base table row gets deleted, any dependent
sparse table row augmentations get also deleted implicitly.

4.4.2.4. The table's reorders Statement for Table Indexing

The table's `reorders' statement is used to supply table indexing
information of tables, that contain exactly the same index objects of
a base table but in a different order. It gets at least two
arguments. The first one specifies the identifier of the base table.
The second one specifies a comma-separated list of exactly those
object identifiers of the base table's `index' statement, but in the
order to be used in this table. Note that a reordered table cannot
itself be reordered. Anyhow, a base table may be used for multiple
reordered tables.

Under some circumstances, an optional `implied' keyword may be added
in front of the list to indicate a compact encoding of the last
object in the list. See Section 2.2 for details.

Instances of subordinate columnar objects of a reordered table exist
according to the same semantics as instances of subordinate columnar
objects of the base table. As such, note that creation of a base
table row implies the correspondent creation of any related reordered
table row. Reordered tables MUST NOT be used in table row creation
and deletion operations.

4.4.2.5. The table's expands Statement for Table Indexing

The table's `expands' statement is used to supply table indexing
information of table expansions. Table expansions use exactly the
same index objects of another table together with additional indexing
objects. Thus, the `expands' statement gets at least two arguments.
The first one specifies the identifier of the base table. The second
one specifies a comma-separated list of the additional object
identifiers used for indexing. Note that an expanded table may
itself be expanded, and base tables may be used for multiple table
expansions.

Under some circumstances, an optional `implied' keyword may be added
in front of the list to indicate a compact encoding of the last
object in the list. See Section 2.2 for details.

4.4.3. The table's create Statement

The table's `create' statement, which need not be present, gets no
argument. If the `create' statement is present, table row creation
(and deletion) is possible.

4.4.4. The table's object Statement

The table's `object' statement, which must be present at least once,
makes this table contain a given columnar object. It gets two
arguments: the name of the columnar object to be defined and a
statement block that holds additional detailed information in an
obligatory order.

4.4.4.1. The object's implements Statement

The `implements' statement, which must be present, is used to specify
a single leaf attribute of a class that is implemented by this
columnar object. The type of this attribute must be a simple type,
i.e., not a class.

4.4.4.2. The object's subid Statement

The `subid' statement, which need not be present, is used to specify
the sub-identifier that identifies the columnar object within this
table, i.e., the object identifier of the columnar object is the
concatenation of the values of this table's oid statement and of this
subid statement.

If this statement is omitted, the sub-identifier is the one of the
previous object statement within this table plus 1. If the
containing object statement is the first one within the containing
table and the subid statement is omitted, the sub-identifier is 1.

4.4.4.3. The object's status Statement

The object's `status' statement, which need not be present, gets one
argument which is used to specify whether this columnar object
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.

Columnar objects SHOULD NOT be defined as `current' if the
implemented attribute definition is `deprecated' or `obsolete'.
Similarly, they SHOULD NOT be defined as `deprecated' if the
implemented attribute is `obsolete'. Nevertheless, subsequent
revisions of used class definitions cannot be avoided, but SHOULD be
taken into account in subsequent revisions of the local module.

Note that it is RECOMMENDED to omit the status statement which means
that the status is inherited from the containing table statement.
However, if the status of a columnar object varies from the
containing table, it has to be expressed explicitly, e.g., if the
implemented attribute has been deprecated or obsoleted.

4.4.4.4. The object's description Statement

The object's `description' statement, which need not be present, gets
one argument which is used to specify a high-level textual
description of this columnar object.

Note that in contrast to other definitions this description statement
is not mandatory and it is RECOMMENDED to omit it, if the object is
fully described by the description of the implemented attribute.

4.4.4.5. The object's reference Statement

The object's `reference' statement, which need not be present, gets
one argument which is used to specify a textual cross-reference to
some other document, either another module which defines related
definitions, or some other document which provides additional
information relevant to this columnar object.

   It is RECOMMENDED to omit this statement, if the object's references
   are fully described by the implemented attribute.

4.4.5. The table's status Statement

   The table's `status' statement, which must be present, gets one
   argument which is used to specify whether this table 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.

4.4.6. The table's description Statement

   The table's `description' statement, which must be present, gets one
   argument which is used to specify a high-level textual description of
   this table.

   It is RECOMMENDED to include all semantic definitions necessary for
   the implementation of this table.

4.4.7. The table's reference Statement

   The table's `reference' statement, which need not be present, gets
   one argument which is used to specify a textual cross-reference to
   some other document, either another module which defines related
   definitions, or some other document which provides additional
   information relevant to this table statement.

4.4.8. Usage Example

```
table ifTable {
  oid             interfaces.2;
  index           (ifIndex);
  object ifIndex { implements Interface.index;       };
  object ifDescr { implements Interface.description; };
  // ...
  status          current;
  description
        "This table implements the Interface class.";
};
```

4.5. The notification Statement

   The `notification' statement is used to map events defined within
   classes to SNMP notifications. The `notification' statement gets two
   arguments: a lower-case notification identifier and a statement block
   that holds detailed notification information in an obligatory order.

   See the `notificationStatement' rule of the grammar (Section 5) for
   the formal syntax of the `notification' statement.

4.5.1. The notification's oid Statement

   The notification's `oid' statement, which must be present, gets one
   argument which specifies the object identifier value that is assigned
   to this notification.

4.5.2. The notification's signals Statement

   The notification's `signals' statement, which must be present,
   denotes the event that is signaled by this notification. The
   statement gets two arguments: the event to be signaled (in the
   qualified form `Class.event') and a statement block that holds
   detailed information on the objects transmitted with this
   notification in an obligatory order.

4.5.2.1. The signals' object Statement

   The signals' `object' statement, which can be present zero, one or
   multiple times, makes a single instance of a class attribute be
   contained in this notification. It gets one argument: the specific
   class attribute. The namespace of attributes not specified by
   qualified names is the namespace of the event's class specified in
   the `signals' statement.

4.5.3. The notification's status Statement

   The notification's `status' statement, which must be present, gets
   one argument which is used to specify whether this notification
   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.

4.5.4. The notification's description Statement

   The notification's `description' statement, which need not be
   present, gets one argument which is used to specify a high-level
   textual description of this notification.

   It is RECOMMENDED to include all semantics and purposes of this
   notification.

4.5.5. The notification's reference Statement

   The notification's `reference' statement, which need not be present,
   gets one argument which is used to specify a textual cross-reference
   to some other document, either another module which defines related
   definitions, or some other document which provides additional
   information relevant to this notification statement.

4.5.6. Usage Example

```
notification linkDown {
    oid         snmpTraps.3;
    signals     Interface.linkDown {
        object      ifIndex;
        object      ifAdminStatus;
        object      ifOperStatus;
    };
    status      current;
    description
            "This notification signals the linkDown event
             of the Interface class.";
};
```

4.6. The group Statement

   The `group' statement is used to define a group of arbitrary nodes in
   the object identifier tree. It gets two arguments: a lower-case
   group identifier and a statement block that holds detailed group
   information in an obligatory order.

   Note that the primary application of groups are compliance
   statements, although they might be referred in other formal or
   informal documents.

   See the `groupStatement' rule of the grammar (Section 5) for the
   formal syntax of the `group' statement.

4.6.1.   The group's oid Statement

   The group's `oid' statement, which must be present, gets one argument
   which specifies the object identifier value that is assigned to this
   group.

4.6.2.   The group's members Statement

   The group's `members' statement, which must be present, gets one
   argument which specifies the list of nodes by their identifiers to be
   contained in this group. The list of nodes has to be comma-separated
   and enclosed in parenthesis.

4.6.3.   The group's status Statement

   The group's `status' statement, which must be present, gets one
   argument which is used to specify whether this group 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 the group should no longer be used. While the value
   `deprecated' also indicates an obsolete definition, it permits
   new/continued use of this group.

4.6.4.   The group's description Statement

   The group's `description' statement, which must be present, gets one
   argument which is used to specify a high-level textual description of
   this group. It is RECOMMENDED to include any relation to other
   groups.

4.6.5.   The group's reference Statement

   The group's `reference' statement, which need not be present, gets
   one argument which is used to specify a textual cross-reference to
   some other document, either another module which defines related
   groups, or some other document which provides additional information
   relevant to this group.

4.6.6. Usage Example

   The snmpGroup, originally defined in [RFC3418], may be described as
   follows:

   group snmpGroup {
     oid   snmpMIBGroups.8;
     objects   (snmpInPkts, snmpInBadVersions,
                       snmpInASNParseErrs,
                       snmpSilentDrops, snmpProxyDrops,

```
                         snmpEnableAuthenTraps);
    status             current;
    description
             "A collection of objects providing basic
              instrumentation and control of an agent.";
   };
```

4.7. The compliance Statement

The `compliance' statement is used to define a set of conformance
requirements, named a `compliance statement'. It gets two arguments:
a lower-case compliance identifier and a statement block that holds
detailed compliance information in an obligatory order.

See the `complianceStatement' rule of the grammar (Section 5) for the
formal syntax of the `compliance' statement.

4.7.1. The compliance's oid Statement

The compliance's `oid' statement, which must be present, gets one
argument which specifies the object identifier value that is assigned
to this compliance statement.

4.7.2. The compliance's status Statement

The compliance's `status' statement, which must be present, gets one
argument which is used to specify whether this compliance statement
is current or historic. The value `current' means that the
definition is current and valid. The value `obsolete' means the
definition is obsolete and no longer specifies a valid definition of
conformance. While the value `deprecated' also indicates an obsolete
definition, it permits new/continued use of the compliance
specification.

4.7.3. The compliance's description Statement

The compliance's `description' statement, which must be present, gets
one argument which is used to specify a high-level textual
description of this compliance statement.

4.7.4. The compliance's reference Statement

The compliance's `reference' statement, which need not be present,
gets one argument which is used to specify a textual cross-reference
to some other document, either another module which defines related
compliance statements, or some other document which provides
additional information relevant to this compliance statement.

4.7.5. The compliance's mandatory Statement

   The compliance's `mandatory' statement, which need not be present,
   gets one argument which is used to specify a comma-separated list of
   one or more groups (Section 4.6) of objects and/or notifications
   enclosed in parenthesis. These groups are unconditionally mandatory
   for implementation.

   If an agent claims compliance to a MIB module then it must implement
   each and every object and notification within each group listed in
   the `mandatory' statement(s) of the compliance statement(s) of that
   module.

4.7.6. The compliance's optional Statement

   The compliance's `optional' statement, which need not be present, is
   repeatedly used to name each group which is conditionally mandatory
   for compliance to the compliance statement. It can also be used to
   name unconditionally optional groups. A group named in an `optional'
   statement MUST be absent from the correspondent `mandatory'
   statement. The `optional' statement gets two arguments: a lower-case
   group identifier and a statement block that holds detailed compliance
   information on that group.

   Conditionally mandatory groups include those groups which are
   mandatory only if a particular protocol is implemented, or only if
   another group is implemented. The `description' statement specifies
   the conditions under which the group is conditionally mandatory.

   A group which is named in neither a `mandatory' statement nor an
   `optional' statement, is unconditionally optional for compliance to
   the module.

   See the `optionalStatement' rule of the grammar (Section 5) for the
   formal syntax of the `optional' statement.

4.7.6.1. The optional's description Statement

   The optional's `description' statement, which must be present, gets
   one argument which is used to specify a high-level textual
   description of the conditions under which this group is conditionally
   mandatory or unconditionally optional.

4.7.7. The compliance's refine Statement

   The compliance's `refine' statement, which need not be present, is
   repeatedly used to specify each object for which compliance has a
   refined requirement with respect to the module definition. The

object must be present in one of the conformance groups named in the
correspondent `mandatory' or `optional' statements. The `refine'
statement gets two arguments: a lower-case identifier of a scalar or
columnar object and a statement block that holds detailed refinement
information on that object.

See the `refineStatement' rule of the grammar (Section 5) for the
formal syntax of the `refine' statement.

4.7.7.1. The refine's type Statement

The refine's `type' statement, which need not be present, gets one
argument that is used to provide a refined type for the correspondent
object. Type restrictions may be applied by appending subtyping
information according to the rules of the base type. See [RFC3780]
for SMIng base types and their type restrictions. In case of
enumeration or bitset types the order of named numbers is not
significant.

Note that if a `type' and a `writetype' statement are both present
then this type only applies when instances of the correspondent
object are read.

4.7.7.2. The refine's writetype Statement

The refine's `writetype' statement, which need not be present, gets
one argument that is used to provide a refined type for the
correspondent object, only when instances of that object are written.
Type restrictions may be applied by appending subtyping information
according to the rules of the base type. See [RFC3780] for SMIng
base types and their type restrictions. In case of enumeration or
bitset types the order of named numbers is not significant.

4.7.7.3. The refine's access Statement

The refine's `access' statement, which need not be present, gets one
argument that is used to specify the minimal level of access that the
correspondent object must implement in the sense of its original
`access' statement.  Hence, the refine's `access' statement MUST NOT
specify a greater level of access than is specified in the
correspondent object definition.

An implementation is compliant if the level of access it provides is
greater or equal to the minimal level in the refine's `access'
statement and less or equal to the maximal level in the object's
`access' statement.

4.7.7.4. The refine's description Statement

The refine's `description' statement, which must be present, gets one
argument which is used to specify a high-level textual description of
the refined compliance requirement.

4.7.8. Usage Example

The compliance statement contained in the SNMPv2-MIB [RFC3418],
converted to SMIng:

```
compliance snmpBasicComplianceRev2 {
  oid              snmpMIBCompliances.3;
  status           current;
  description
          "The compliance statement for SNMP entities which
           implement this MIB module.";

  mandatory        (snmpGroup, snmpSetGroup, systemGroup,
                    snmpBasicNotificationsGroup);

  optional snmpCommunityGroup {
    description
          "This group is mandatory for SNMP entities which
           support community-based authentication.";
  };
  optional snmpWarmStartNotificationGroup {
    description
          "This group is mandatory for an SNMP entity which
           supports command responder applications, and is
           able to reinitialize itself such that its
           configuration is unaltered.";
  };
};
```

5. NMRG-SMING-SNMP-EXT

The grammar of the snmp statement (including all its contained
statements) conforms to the Augmented Backus-Naur Form (ABNF)
[RFC2234]. It is included in the abnf statement of the snmp SMIng
extension definition in the NMRG-SMING-SNMP-EXT module below.

```
module NMRG-SMING-SNMP-EXT {

  organization     "IRTF Network Management Research Group (NMRG)";

  contact          "IRTF Network Management Research Group (NMRG)
                    http://www.ibr.cs.tu-bs.de/projects/nmrg/
```

                        Frank Strauss
                        TU Braunschweig
                        Muehlenpfordtstrasse 23
                        38106 Braunschweig
                        Germany
                        Phone: +49 531 391 3266
                        EMail: strauss@ibr.cs.tu-bs.de

                        Juergen Schoenwaelder
                        International University Bremen
                        P.O. Box 750 561
                        28725 Bremen
                        Germany
                        Phone: +49 421 200 3587
                        EMail: j.schoenwaelder@iu-bremen.de";

        description   "This module defines a SMIng extension to define
                        the mapping of SMIng definitions of class and
                        their attributes and events to SNMP compatible
                        definitions of modules, node, scalars, tables,
                        and notifications, and additional information on
                        module compliances.

                        Copyright (C) The Internet Society (2004).
                        All Rights Reserved.
                        This version of this module is part of
                        RFC 3781, see the RFC itself for full
                        legal notices.";

        revision {
            date   "2003-12-16";
            description "Initial revision, published as RFC 3781.";
        };

        //
        //
        //

        extension snmp {

            status   current;
            description
               "The snmp statement maps SMIng definitions to SNMP
                conformant definitions.";
            abnf "
  ;;
  ;; sming-snmp.abnf -- Grammar of SNMP mappings in ABNF
  ;;   notation (RFC 2234).

```
;;
;; @(#) $Id: sming-snmp.abnf,v 1.14 2003/10/23 19:31:55 strauss Exp $
;;
;; Copyright (C) The Internet Society (2004). All Rights Reserved.
;;

;;
;; Statement rules.
;;
snmpStatement           = snmpKeyword *1(sep lcIdentifier) optsep
                                \"{\" stmtsep
                                *1(oidStatement stmtsep)
                                *(nodeStatement stmtsep)
                                *(scalarsStatement stmtsep)
                                *(tableStatement stmtsep)
                                *(notificationStatement stmtsep)
                                *(groupStatement stmtsep)
                                *(complianceStatement stmtsep)
                                statusStatement stmtsep
                                descriptionStatement stmtsep
                                *1(referenceStatement stmtsep)
                          \"}\" optsep \";\"

nodeStatement           = nodeKeyword sep lcIdentifier optsep
                                \"{\" stmtsep
                                oidStatement stmtsep
                                *1(representsStatement stmtsep)
                                statusStatement stmtsep
                                *1(descriptionStatement stmtsep)
                                *1(referenceStatement stmtsep)
                          \"}\" optsep \";\"

representsStatement     = representsKeyword sep
                                qucIdentifier optsep \";\"

scalarsStatement        = scalarsKeyword sep lcIdentifier optsep
                                \"{\" stmtsep
                                oidStatement stmtsep
                                1*(objectStatement stmtsep)
                                statusStatement stmtsep
                                descriptionStatement stmtsep
                                *1(referenceStatement stmtsep)
                          \"}\" optsep \";\"

tableStatement          = tableKeyword sep lcIdentifier optsep
                                \"{\" stmtsep
                                oidStatement stmtsep
```

```
                           anyIndexStatement stmtsep
                           *1(createStatement stmtsep)
                           1*(objectStatement stmtsep)
                           statusStatement stmtsep
                           descriptionStatement stmtsep
                           *1(referenceStatement stmtsep)
                       \"}\" optsep \";\"

  objectStatement       = objectKeyword sep lcIdentifier optsep
                           \"{\" stmtsep
                           implementsStatement stmtsep
                           *1(subidStatement stmtsep)
                           *1(statusStatement stmtsep)
                           *1(descriptionStatement stmtsep)
                           *1(referenceStatement stmtsep)
                       \"}\" optsep \";\"

  implementsStatement   = implementsKeyword sep qcattrIdentifier
                           optsep \";\"

  notificationStatement = notificationKeyword sep lcIdentifier
                           optsep \"{\" stmtsep
                           oidStatement stmtsep
                           signalsStatement stmtsep
                           statusStatement stmtsep
                           descriptionStatement stmtsep
                           *1(referenceStatement stmtsep)
                       \"}\" optsep \";\"

  signalsStatement      = signalsKeyword sep qattrIdentifier
                           optsep \"{\" stmtsep
                           *(signalsObjectStatement)
                       \"}\" optsep \";\"

  signalsObjectStatement = objectKeyword sep
                           qattrIdentifier optsep \";\"

  groupStatement        = groupKeyword sep lcIdentifier optsep
                           \"{\" stmtsep
                           oidStatement stmtsep
                           membersStatement stmtsep
                           statusStatement stmtsep
                           descriptionStatement stmtsep
                           *1(referenceStatement stmtsep)
                       \"}\" optsep \";\"

  complianceStatement   = complianceKeyword sep lcIdentifier optsep
                           \"{\" stmtsep
```

```
                             oidStatement stmtsep
                             statusStatement stmtsep
                             descriptionStatement stmtsep
                             *1(referenceStatement stmtsep)
                             *1(mandatoryStatement stmtsep)
                             *(optionalStatement stmtsep)
                             *(refineStatement stmtsep)
                      \"}\" optsep \";\"

 anyIndexStatement          = indexStatement /
                             augmentsStatement /
                             reordersStatement /
                             extendsStatement /
                             expandsStatement

 indexStatement             = indexKeyword *1(sep impliedKeyword) optsep
                             \"(\" optsep qlcIdentifierList
                             optsep \")\" optsep \";\"

 augmentsStatement          = augmentsKeyword sep qlcIdentifier
                             optsep \";\"

 reordersStatement          = reordersKeyword sep qlcIdentifier
                             *1(sep impliedKeyword)
                             optsep \"(\" optsep
                             qlcIdentifierList optsep \")\"
                             optsep \";\"

 extendsStatement           = extendsKeyword sep qlcIdentifier optsep \";\"

 expandsStatement           = expandsKeyword sep qlcIdentifier
                             *1(sep impliedKeyword)
                             optsep \"(\" optsep
                             qlcIdentifierList optsep \")\"
                             optsep \";\"

 createStatement            = createKeyword optsep \";\"

 membersStatement           = membersKeyword optsep \"(\" optsep
                             qlcIdentifierList optsep
                             \")\" optsep \";\"

 mandatoryStatement         = mandatoryKeyword optsep \"(\" optsep
                             qlcIdentifierList optsep
                             \")\" optsep \";\"

 optionalStatement          = optionalKeyword sep qlcIdentifier optsep
                             \"{\" descriptionStatement stmtsep
```

```
                              \"}\" optsep \";\"

  refineStatement              = refineKeyword sep qlcIdentifier optsep \"{\"
                                    *1(typeStatement stmtsep)
                                    *1(writetypeStatement stmtsep)
                                    *1(accessStatement stmtsep)
                                    descriptionStatement stmtsep
                                 \"}\" optsep \";\"

  typeStatement                = typeKeyword sep
                                    (refinedBaseType / refinedType)
                                    optsep \";\"

  writetypeStatement           = writetypeKeyword sep
                                    (refinedBaseType / refinedType)
                                    optsep \";\"

  oidStatement                 = oidKeyword sep objectIdentifier optsep \";\"

  subidStatement               = subidKeyword sep subid optsep \";\"

  ;;
  ;; Statement keywords.
  ;;

  snmpKeyword          = %x73 %x6E %x6D %x70
  nodeKeyword          = %x6E %x6F %x64 %x65
  representsKeyword    = %x72 %x65 %x70 %x72 %x65 %x73 %x65 %x6E %x74
                         %x73
  scalarsKeyword       = %x73 %x63 %x61 %x6C %x61 %x72 %x73
  tableKeyword         = %x74 %x61 %x62 %x6C %x65
  implementsKeyword    = %x69 %x6D %x70 %x6C %x65 %x6D %x65 %x6E %x74
                         %x73
  subidKeyword         = %x73 %x75 %x62 %x69 %x64
  objectKeyword        = %x6F %x62 %x6A %x65 %x63 %x74
  notificationKeyword  = %x6E %x6F %x74 %x69 %x66 %x69 %x63 %x61 %x74
                         %x69 %x6F %x6E
  signalsKeyword       = %x73 %x69 %x67 %x6E %x61 %x6C %x73
  oidKeyword           = %x6F %x69 %x64
  groupKeyword         = %x67 %x72 %x6F %x75 %x70
  complianceKeyword    = %x63 %x6F %x6D %x70 %x6C %x69 %x61 %x6E %x63
                         %x65
  impliedKeyword       = %x69 %x6D %x70 %x6C %x69 %x65 %x64
  indexKeyword         = %x69 %x6E %x64 %x65 %x78
  augmentsKeyword      = %x61 %x75 %x67 %x6D %x65 %x6E %x74 %x73
  reordersKeyword      = %x72 %x65 %x6F %x72 %x64 %x65 %x72 %x73
  extendsKeyword       = %x65 %x78 %x74 %x65 %x6E %x64 %x73
  expandsKeyword       = %x65 %x78 %x70 %x61 %x6E %x64 %x73
```

```
createKeyword         = %x63 %x72 %x65 %x61 %x74 %x65
membersKeyword        = %x6D %x65 %x6D %x62 %x65 %x72 %x73
mandatoryKeyword      = %x6D %x61 %x6E %x64 %x61 %x74 %x6F %x72 %x79
optionalKeyword       = %x6F %x70 %x74 %x69 %x6F %x6E %x61 %x6C
refineKeyword         = %x72 %x65 %x66 %x69 %x6E %x65
writetypeKeyword      = %x77 %x72 %x69 %x74 %x65 %x74 %x79 %x70 %x65

;; End of ABNF
             ";
    };
    //
    //
    //

    snmp {

        node ccitt                       { oid 0;          };

        node   zeroDotZero {
            oid         0.0;
            description "A null value used for pointers.";
        };

        node iso                         { oid 1;          };
        node    org                      { oid iso.3;      };
        node       dod                   { oid org.6;      };
        node         internet            { oid dod.1;      };
        node           directory         { oid internet.1; };
        node           mgmt              { oid internet.2; };
        node             mib-2           { oid mgmt.1;     };
        node               transmission  { oid mib-2.10;   };
        node           experimental      { oid internet.3; };
        node           private           { oid internet.4; };
        node             enterprises     { oid private.1;  };
        node           security          { oid internet.5; };
        node           snmpV2            { oid internet.6; };
        node             snmpDomains     { oid snmpV2.1;   };
        node             snmpProxys      { oid snmpV2.2;   };
        node             snmpModules     { oid snmpV2.3;   };

        node joint-iso-ccitt             { oid 2;          };

        status          current;
        description
           "This set of nodes defines the core object
            identifier hierarchy";
        reference
           "RFC 2578, Section 2.";
```

```
        };

   };

6.  NMRG-SMING-SNMP

    The module NMRG-SMING-SNMP specified below defines derived types that
    are specific to the SNMP mapping.

module NMRG-SMING-SNMP {

        organization      "IRTF Network Management Research Group (NMRG)";

        contact           "IRTF Network Management Research Group (NMRG)
                           http://www.ibr.cs.tu-bs.de/projects/nmrg/

                           Frank Strauss
                           TU Braunschweig
                           Muehlenpfordtstrasse 23
                           38106 Braunschweig
                           Germany
                           Phone: +49 531 391 3266
                           EMail: strauss@ibr.cs.tu-bs.de

                           Juergen Schoenwaelder
                           International University Bremen
                           P.O. Box 750 561
                           28725 Bremen
                           Germany
                           Phone: +49 421 200 3587
                           EMail: j.schoenwaelder@iu-bremen.de";

        description       "Core type definitions for the SMIng SNMP mapping.
                           These definitions are based on RFC 2579 definitions
                           that are specific to the SNMP protocol and its
                           naming system.

                           Copyright (C) The Internet Society (2004).
                           All Rights Reserved.
                           This version of this module is part of
                           RFC 3781, see the RFC itself for full
                           legal notices.";

        revision {
            date          "2003-12-16";
            description "Initial version, published as RFC 3781.";
        };
```

```
    typedef TestAndIncr {
        type        Integer32 (0..2147483647);
        description
            "Represents integer-valued information used for atomic
             operations.  When the management protocol is used to
             specify that an object instance having this type is to
             be modified, the new value supplied via the management
             protocol must precisely match the value presently held by
             the instance.  If not, the management protocol set
             operation fails with an error of `inconsistentValue'.
             Otherwise, if the current value is the maximum value of
             2^31-1 (2147483647 decimal), then the value held by the
             instance is wrapped to zero; otherwise, the value held by
             the instance is incremented by one.  (Note that
             regardless of whether the management protocol set
             operation succeeds, the variable-binding in the request
             and response PDUs are identical.)

             The value of the SNMP access clause for objects having
             this type has to be `readwrite'.  When an instance of a
             columnar object having this type is created, any value
             may be supplied via the management protocol.

             When the network management portion of the system is re-
             initialized, the value of every object instance having
             this type must either be incremented from its value prior
             to the re-initialization, or (if the value prior to the
             re-initialization is unknown) be set to a
             pseudo-randomly generated value."; };

    typedef AutonomousType {
        type        Pointer;
        description
            "Represents an independently extensible type
             identification value.  It may, for example, indicate a
             particular OID sub-tree with further MIB definitions, or
             define a particular type of protocol or hardware.";
    };

    typedef VariablePointer {
        type        Pointer;
        description
            "A pointer to a specific object instance.  For example,
             sysContact.0 or ifInOctets.3.";
    };

    typedef RowPointer {
        type        Pointer;
```

```
        description
            "Represents a pointer to a conceptual row.  The value is
            the name of the instance of the first accessible columnar
            object in the conceptual row.

            For example, ifIndex.3 would point to the 3rd row in the
            ifTable (note that if ifIndex were not-accessible, then
            ifDescr.3 would be used instead).";
    };

    typedef RowStatus {
        type          Enumeration (active(1), notInService(2),
                          notReady(3), createAndGo(4),
                          createAndWait(5), destroy(6));
        description
        "The RowStatus type is used to manage the creation and
         deletion of conceptual rows, and is used as the type for the
         row status column of a conceptual row.

         The status column has six defined values:

            - `active', which indicates that the conceptual row is
            available for use by the managed device;

            - `notInService', which indicates that the conceptual
            row exists in the agent, but is unavailable for use by
            the managed device (see NOTE below);

            - `notReady', which indicates that the conceptual row
            exists in the agent, but is missing information
            necessary in order to be available for use by the
            managed device;

            - `createAndGo', which is supplied by a management
            station wishing to create a new instance of a
            conceptual row and to have its status automatically set
            to active, making it available for use by the managed
            device;

            - `createAndWait', which is supplied by a management
            station wishing to create a new instance of a
            conceptual row (but not make it available for use by
            the managed device); and,

            - `destroy', which is supplied by a management station
            wishing to delete all of the instances associated with
            an existing conceptual row.
```

Whereas five of the six values (all except `notReady') may
be specified in a management protocol set operation, only
three values will be returned in response to a management
protocol retrieval operation: `notReady', `notInService' or
`active'. That is, when queried, an existing conceptual row
has only three states: it is either available for use by the
managed device (the status column has value `active'); it is
not available for use by the managed device, though the
agent has sufficient information to make it so (the status
column has value `notInService'); or, it is not available
for use by the managed device, and an attempt to make it so
would fail because the agent has insufficient information
(the state column has value `notReady').

                        NOTE WELL

This textual convention may be used for a MIB table,
irrespective of whether the values of that table's
conceptual rows are able to be modified while it is
active, or whether its conceptual rows must be taken
out of service in order to be modified. That is, it is
the responsibility of the DESCRIPTION clause of the
status column to specify whether the status column must
not be `active' in order for the value of some other
column of the same conceptual row to be modified. If
such a specification is made, affected columns may be
changed by an SNMP set PDU if the RowStatus would not
be equal to `active' either immediately before or after
processing the PDU. In other words, if the PDU also
contained a varbind that would change the RowStatus
value, the column in question may be changed if the
RowStatus was not equal to `active' as the PDU was
received, or if the varbind sets the status to a value
other than 'active'.

Also note that whenever any elements of a row exist, the
RowStatus column must also exist.

To summarize the effect of having a conceptual row with a
column having a type of RowStatus, consider the following
state diagram:

                                     STATE
             +---------------+-----------+-------------+-------------
             | A | B | C | D
             | status column |status col.|status column|
             | | is | is | status column
      ACTION |does not exist | notReady | notInService| is active
-------------+---------------+-----------+-------------+-------------
set status |noError   ->D |inconsist- |inconsistent-|inconsistent-
column to | or | entValue | Value | Value
createAndGo |inconsistent- | | |
             | Value | | |
-------------+---------------+-----------+-------------+-------------
set status |noError see 1 |inconsist- |inconsistent-|inconsistent-
column to | or | entValue | Value | Value
createAndWait|wrongValue | | |
-------------+---------------+-----------+-------------+-------------
set status |inconsistent- |inconsist- |noError |noError
column to | Value | entValue | |
active | | | |
             | | or | |
             | | | |
             | |see 2 ->D |see 8   ->D | ->D
-------------+---------------+-----------+-------------+-------------
set status |inconsistent- |inconsist- |noError |noError   ->C
column to | Value | entValue | |
notInService | | | |
             | | or | | or
             | | | |
             | |see 3 ->C | ->C |see 6
-------------+---------------+-----------+-------------+-------------
set status |noError |noError |noError |noError   ->A
column to | | | | or
destroy | ->A | ->A | ->A |see 7
-------------+---------------+-----------+-------------+-------------
set any other|see 4 |noError |noError |see 5
column to some| | | |
value | | see 1 | ->C | ->D
-------------+---------------+-----------+-------------+-------------

      (1) go to B or C, depending on information available to the
      agent.

      (2) if other variable bindings included in the same PDU,
      provide values for all columns which are missing but
      required, then return noError and goto D.

(3) if other variable bindings included in the same PDU,
provide values for all columns which are missing but
required, then return noError and goto C.

(4) at the discretion of the agent, the return value may be
either:

    inconsistentName: because the agent does not choose to
    create such an instance when the corresponding
    RowStatus instance does not exist, or

    inconsistentValue: if the supplied value is
    inconsistent with the state of some other MIB object's
    value, or

    noError: because the agent chooses to create the
    instance.

If noError is returned, then the instance of the status
column must also be created, and the new state is B or C,
depending on the information available to the agent. If
inconsistentName or inconsistentValue is returned, the row
remains in state A.

(5) depending on the MIB definition for the column/table,
either noError or inconsistentValue may be returned.

(6) the return value can indicate one of the following
errors:

    wrongValue: because the agent does not support
    createAndWait, or

    inconsistentValue: because the agent is unable to take
    the row out of service at this time, perhaps because it
    is in use and cannot be de-activated.

(7) the return value can indicate the following error:

    inconsistentValue: because the agent is unable to
    remove the row at this time, perhaps because it is in
    use and cannot be de-activated.

NOTE: Other processing of the set request may result in a
response other than noError being returned, e.g.,
wrongValue, noCreation, etc.

Conceptual Row Creation

There are four potential interactions when creating a
conceptual row: selecting an instance-identifier which is
not in use; creating the conceptual row; initializing any
objects for which the agent does not supply a default; and,
making the conceptual row available for use by the managed
device.

Interaction 1: Selecting an Instance-Identifier

The algorithm used to select an instance-identifier varies
for each conceptual row. In some cases, the instance-
identifier is semantically significant, e.g., the
destination address of a route, and a management station
selects the instance-identifier according to the semantics.

In other cases, the instance-identifier is used solely to
distinguish conceptual rows, and a management station
without specific knowledge of the conceptual row might
examine the instances present in order to determine an
unused instance-identifier. (This approach may be used, but
it is often highly sub-optimal; however, it is also a
questionable practice for a naive management station to
attempt conceptual row creation.)

Alternately, the MIB module which defines the conceptual row
might provide one or more objects which provide assistance
in determining an unused instance-identifier. For example,
if the conceptual row is indexed by an integer-value, then
an object having an integer-valued SYNTAX clause might be
defined for such a purpose, allowing a management station to
issue a management protocol retrieval operation. In order
to avoid unnecessary collisions between competing management
stations, `adjacent' retrievals of this object should be
different.

Finally, the management station could select a pseudo-random
number to use as the index. In the event that this index
was already in use and an inconsistentValue was returned in
response to the management protocol set operation, the
management station should simply select a new pseudo-random
number and retry the operation.

A MIB designer should choose between the two latter
algorithms based on the size of the table (and therefore the
efficiency of each algorithm). For tables in which a large
number of entries are expected, it is recommended that a MIB

object be defined that returns an acceptable index for
creation. For tables with small numbers of entries, it is
recommended that the latter pseudo-random index mechanism be
used.

Interaction 2: Creating the Conceptual Row

Once an unused instance-identifier has been selected, the
management station determines if it wishes to create and
activate the conceptual row in one transaction or in a
negotiated set of interactions.

Interaction 2a: Creating and Activating the Conceptual Row

The management station must first determine the column
requirements, i.e., it must determine those columns for
which it must or must not provide values. Depending on the
complexity of the table and the management station's
knowledge of the agent's capabilities, this determination
can be made locally by the management station. Alternately,
the management station issues a management protocol get
operation to examine all columns in the conceptual row that
it wishes to create. In response, for each column, there
are three possible outcomes:

   - a value is returned, indicating that some other
   management station has already created this conceptual
   row. We return to interaction 1.

   - the exception `noSuchInstance' is returned,
   indicating that the agent implements the object-type
   associated with this column, and that this column in at
   least one conceptual row would be accessible in the MIB
   view used by the retrieval were it to exist. For those
   columns to which the agent provides read-create access,
   the `noSuchInstance' exception tells the management
   station that it should supply a value for this column
   when the conceptual row is to be created.

   - the exception `noSuchObject' is returned, indicating
   that the agent does not implement the object-type
   associated with this column or that there is no
   conceptual row for which this column would be
   accessible in the MIB view used by the retrieval. As
   such, the management station can not issue any
   management protocol set operations to create an
   instance of this column.

Once the column requirements have been determined, a
management protocol set operation is accordingly issued.
This operation also sets the new instance of the status
column to `createAndGo'.

When the agent processes the set operation, it verifies that
it has sufficient information to make the conceptual row
available for use by the managed device. The information
available to the agent is provided by two sources: the
management protocol set operation which creates the
conceptual row, and, implementation-specific defaults
supplied by the agent (note that an agent must provide
implementation-specific defaults for at least those objects
which it implements as read-only). If there is sufficient
information available, then the conceptual row is created, a
`noError' response is returned, the status column is set to
`active', and no further interactions are necessary (i.e.,
interactions 3 and 4 are skipped). If there is insufficient
information, then the conceptual row is not created, and the
set operation fails with an error of `inconsistentValue'.
On this error, the management station can issue a management
protocol retrieval operation to determine if this was
because it failed to specify a value for a required column,
or, because the selected instance of the status column
already existed. In the latter case, we return to
interaction 1. In the former case, the management station
can re-issue the set operation with the additional
information, or begin interaction 2 again using
`createAndWait' in order to negotiate creation of the
conceptual row.

                        NOTE WELL

    Regardless of the method used to determine the column
    requirements, it is possible that the management
    station might deem a column necessary when, in fact,
    the agent will not allow that particular columnar
    instance to be created or written. In this case, the
    management protocol set operation will fail with an
    error such as `noCreation' or `notWritable'. In this
    case, the management station decides whether it needs
    to be able to set a value for that particular columnar
    instance. If not, the management station re-issues the
    management protocol set operation, but without setting

a value for that particular columnar instance;
otherwise, the management station aborts the row
creation algorithm.

Interaction 2b: Negotiating the Creation of the Conceptual
Row

The management station issues a management protocol set
operation which sets the desired instance of the status
column to `createAndWait'. If the agent is unwilling to
process a request of this sort, the set operation fails with
an error of `wrongValue'. (As a consequence, such an agent
must be prepared to accept a single management protocol set
operation, i.e., interaction 2a above, containing all of the
columns indicated by its column requirements.) Otherwise,
the conceptual row is created, a `noError' response is
returned, and the status column is immediately set to either
`notInService' or `notReady', depending on whether it has
sufficient information to make the conceptual row available
for use by the managed device. If there is sufficient
information available, then the status column is set to
`notInService'; otherwise, if there is insufficient
information, then the status column is set to `notReady'.
Regardless, we proceed to interaction 3.

Interaction 3: Initializing non-defaulted Objects

The management station must now determine the column
requirements. It issues a management protocol get operation
to examine all columns in the created conceptual row. In
the response, for each column, there are three possible
outcomes:

    - a value is returned, indicating that the agent
    implements the object-type associated with this column
    and had sufficient information to provide a value. For
    those columns to which the agent provides read-create
    access (and for which the agent allows their values to
    be changed after their creation), a value return tells
    the management station that it may issue additional
    management protocol set operations, if it desires, in
    order to change the value associated with this column.

    - the exception `noSuchInstance' is returned,
    indicating that the agent implements the object-type
    associated with this column, and that this column in at
    least one conceptual row would be accessible in the MIB
    view used by the retrieval were it to exist. However,

the agent does not have sufficient information to
provide a value, and until a value is provided, the
conceptual row may not be made available for use by the
managed device. For those columns to which the agent
provides read-create access, the `noSuchInstance'
exception tells the management station that it must
issue additional management protocol set operations, in
order to provide a value associated with this column.

- the exception `noSuchObject' is returned, indicating
that the agent does not implement the object-type
associated with this column or that there is no
conceptual row for which this column would be
accessible in the MIB view used by the retrieval. As
such, the management station can not issue any
management protocol set operations to create an
instance of this column.

If the value associated with the status column is
`notReady', then the management station must first deal with
all `noSuchInstance' columns, if any. Having done so, the
value of the status column becomes `notInService', and we
proceed to interaction 4.

Interaction 4: Making the Conceptual Row Available

Once the management station is satisfied with the values
associated with the columns of the conceptual row, it issues
a management protocol set operation to set the status column
to `active'. If the agent has sufficient information to
make the conceptual row available for use by the managed
device, the management protocol set operation succeeds (a
`noError' response is returned). Otherwise, the management
protocol set operation fails with an error of
`inconsistentValue'.

                       NOTE WELL

A conceptual row having a status column with value
`notInService' or `notReady' is unavailable to the
managed device. As such, it is possible for the
managed device to create its own instances during the
time between the management protocol set operation
which sets the status column to `createAndWait' and the
management protocol set operation which sets the status
column to `active'. In this case, when the management
protocol set operation is issued to set the status
column to `active', the values held in the agent

supersede those used by the managed device.

If the management station is prevented from setting the
status column to `active' (e.g., due to management station or
network failure) the conceptual row will be left in the
`notInService' or `notReady' state, consuming resources
indefinitely. The agent must detect conceptual rows that
have been in either state for an abnormally long period of
time and remove them. It is the responsibility of the
DESCRIPTION clause of the status column to indicate what an
abnormally long period of time would be. This period of time
should be long enough to allow for human response time
(including `think time') between the creation of the
conceptual row and the setting of the status to `active'. In
the absence of such information in the DESCRIPTION clause, it
is suggested that this period be approximately 5 minutes in
length. This removal action applies not only to newly-
created rows, but also to previously active rows which are
set to, and left in, the notInService state for a prolonged
period exceeding that which is considered normal for such a
conceptual row.

Conceptual Row Suspension

When a conceptual row is `active', the management station
may issue a management protocol set operation which sets the
instance of the status column to `notInService'. If the
agent is unwilling to do so, the set operation fails with an
error of `wrongValue' or `inconsistentValue'.
Otherwise, the conceptual row is taken out of service, and a
`noError' response is returned. It is the responsibility of
the DESCRIPTION clause of the status column to indicate
under what circumstances the status column should be taken
out of service (e.g., in order for the value of some other
column of the same conceptual row to be modified).

Conceptual Row Deletion

For deletion of conceptual rows, a management protocol set
operation is issued which sets the instance of the status
column to `destroy'. This request may be made regardless of
the current value of the status column (e.g., it is possible
to delete conceptual rows which are either `notReady',
`notInService' or `active'.) If the operation succeeds, then
all instances associated with the conceptual row are
immediately removed.";
    };

```
    typedef StorageType {
        type           Enumeration (other(1), volatile(2),
                          nonVolatile(3), permanent(4),
                          readOnly(5));
        description
            "Describes the memory realization of a conceptual row.  A
             row which is volatile(2) is lost upon reboot.  A row
             which is either nonVolatile(3), permanent(4) or
             readOnly(5), is backed up by stable storage.  A row which
             is permanent(4) can be changed but not deleted.  A row
             which is readOnly(5) cannot be changed nor deleted.

             If the value of an object with this syntax is either
             permanent(4) or readOnly(5), it cannot be modified.
             Conversely, if the value is either other(1), volatile(2)
             or nonVolatile(3), it cannot be modified to be
             permanent(4) or readOnly(5).  (All illegal modifications
             result in a 'wrongValue' error.)

             Every usage of this textual convention is required to
             specify the columnar objects which a permanent(4) row
             must at a minimum allow to be writable.";
    };

    typedef TDomain {
        type           Pointer;
        description
            "Denotes a kind of transport service.

             Some possible values, such as snmpUDPDomain, are defined
             in the SNMPv2-TM MIB module.  Other possible values are
             defined in other MIB modules."
        reference
            "The SNMPv2-TM MIB module is defined in RFC 3417."
    };

    typedef TAddressOrZero {
        type           OctetString (0..255);
        description
            "Denotes a transport service address.

             A TAddress value is always interpreted within the context
             of a TDomain value.  Thus, each definition of a TDomain
             value must be accompanied by a definition of a textual
             convention for use with that TDomain.  Some possible
             textual conventions, such as SnmpUDPAddress for
             snmpUDPDomain, are defined in the SNMPv2-TM MIB module.
             Other possible textual conventions are defined in other
```

```
            MIB modules.

            A zero-length TAddress value denotes an unknown transport
            service address."
        reference
            "The SNMPv2-TM MIB module is defined in RFC 3417."
    };

    typedef TAddress {
        type        TAddressOrZero (1..255);
        description
            "Denotes a transport service address.

            This type does not allow a zero-length TAddress value."
    };

};
```

7. Security Considerations

    This document presents an extension of the SMIng data definition
    language which supports the mapping of SMIng data definitions so that
    they can be used with the SNMP management framework. The language
    extension and the mapping itself has no security impact on the
    Internet.

8. Acknowledgements

    Since SMIng started as a close successor of SMIv2, some paragraphs
    and phrases are directly taken from the SMIv2 specifications
    [RFC2578], [RFC2579], [RFC2580] written by Jeff Case, Keith
    McCloghrie, David Perkins, Marshall T. Rose, Juergen Schoenwaelder,
    and Steven L. Waldbusser.

    The authors would like to thank all participants of the 7th NMRG
    meeting held in Schloss Kleinheubach from 6-8 September 2000, which
    was a major step towards the current status of this memo, namely
    Heiko Dassow, David Durham, Keith McCloghrie, and Bert Wijnen.

    Furthermore, several discussions within the SMING Working Group
    reflected experience with SMIv2 and influenced this specification at
    some points.

9. References

9.1. Normative References

[RFC3780]   Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation
            Structure of Management Information", RFC 3780, May 2004.

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234]   Crocker, D. and P. Overell, "Augmented BNF for Syntax
            Specifications: ABNF", RFC 2234, November 1997.

9.2. Informative References

[RFC3410]   Case, J., Mundy, R., Partain, D. and B. Stewart,
            "Introduction and Applicability Statements for Internet
            Standard Management Framework", RFC 3410, December 2002.

[RFC2578]   McCloghrie, K., Perkins, D. and J. Schoenwaelder,
            "Structure of Management Information Version 2 (SMIv2)",
            STD 58, RFC 2578, April 1999.

[RFC2579]   McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Textual
            Conventions for SMIv2", STD 59, RFC 2579, April 1999.

[RFC2580]   McCloghrie, K., Perkins, D. and J. Schoenwaelder,
            "Conformance Statements for SMIv2", STD 60, RFC 2580,
            April 1999.

[ASN1]   International Organization for Standardization,
            "Specification of Abstract Syntax Notation One (ASN.1)",
            International Standard 8824, December 1987.

[RFC3159]   McCloghrie, K., Fine, M., Seligson, J., Chan, K., Hahn,
            S., Sahita, R., Smith, A. and F. Reichmeyer, "Structure of
            Policy Provisioning Information (SPPI)", RFC 3159, August
            2001.

[IEEE754]   Institute of Electrical and Electronics Engineers, "IEEE
            Standard for Binary Floating-Point Arithmetic", ANSI/IEEE
            Standard 754-1985, August 1985.

   [RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S.
              Waldbusser, "Management Information Base (MIB) for the
              Simple Network Management Protocol (SNMP)", STD 62, RFC
              3418, December 2002.

   [RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S.
              Waldbusser, "Version 2 of the Protocol Operations for the
              Simple Network Management Protocol (SNMP)", STD 62, RFC
              3416, December 2002.

Authors' Addresses

   Frank Strauss
   TU Braunschweig
   Muehlenpfordtstrasse 23
   38106 Braunschweig
   Germany

   Phone: +49 531 391 3266
   EMail: strauss@ibr.cs.tu-bs.de
   URI: http://www.ibr.cs.tu-bs.de/

   Juergen Schoenwaelder
   International University Bremen
   P.O. Box 750 561
   28725 Bremen
   Germany

   Phone: +49 421 200 3587
   EMail: j.schoenwaelder@iu-bremen.de
   URI: http://www.eecs.iu-bremen.de/

Full Copyright Statement

   Copyright (C) The Internet Society (2004). This document is subject
   to the rights, licenses and restrictions contained in BCP 78, and
   except as set forth therein, the authors retain all their rights.

   This document and the information contained herein are provided on an
   "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS
   OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET
   ENGINEERING TASK FORCE DISCLAIM 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.

Intellectual Property

   The IETF takes no position regarding the validity or scope of any
   Intellectual Property Rights or other rights that might be claimed to
   pertain to the implementation or use of the technology described in
   this document or the extent to which any license under such rights
   might or might not be available; nor does it represent that it has
   made any independent effort to identify any such rights. Information
   on the procedures with respect to rights in RFC documents can be
   found in BCP 78 and BCP 79.

   Copies of IPR disclosures made to the IETF Secretariat and any
   assurances of licenses to be made available, or the result of an
   attempt made to obtain a general license or permission for the use of
   such proprietary rights by implementers or users of this
   specification can be obtained from the IETF on-line IPR repository at
   http://www.ietf.org/ipr.

   The IETF invites any interested party to bring to its attention any
   copyrights, patents or patent applications, or other proprietary
   rights that may cover technology that may be required to implement
   this standard. Please address the information to the IETF at ietf-
   ipr@ietf.org.

Acknowledgement

   Funding for the RFC Editor function is currently provided by the
   Internet Society.