Internet Draft

Andy Bierman
Cisco Systems, Inc.
14 May 2002

Structure of Management Information: Data Structures

<draft-bierman-sming-ds-03.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026 [RFC2026].

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this document is unlimited. Please send comments to the SMIng WG mailing list <sming@ops.ietf.org>.

1. Copyright Notice

2. Abstract

This memo defines a portion of the Structure of Management Information (SMI) for use with network management protocols in the Internet community. In particular, it describes a new structure and naming scheme for network management information, allowing the specification of arbitrarily complex hierarchical data structures.

3. Table of Contents

1 Copyright Notice 1
2 Abstract 2
3 Table of Contents 2
4 The SNMP Network Management Framework 3
5 Overview 4
5.1 Terms 4
5.2 Design Objectives 5
5.3 Data Structure Constructs 6
5.4 Relationship to SMIv2 7
5.5 Hierarchical Instance Naming 8
5.6 SMI-DS Data Object Usage Examples 10
5.6.1 InetAddress Example 10
5.6.2 Generic High Capacity Counter Example 13
5.6.3 Converted SMIv2 TABLE Example 15
5.7 Data Structure Augmentations 17
5.8 SYNTAX POINTER Clause 24
6 Definitions 26
6.1 Namespaces 26
6.2 Syntax 26
7 Information Modules 33
8 Appendix A: SMIv2 Compatibility 34
8.1 Common Constructs 34
8.2 SMIv2 to SMI-DS Module Conversion 34
8.3 SMI-DS to SMIv2 Module Conversion 41
8.4 Compatibility Guidelines 41
9 Appendix B: Complete MODULE Example 42
10 Appendix C: Open Issues 49
11 Appendix D: Discussion of SMIng Objectives 51
12 Security Considerations 66
13 Intellectual Property 67
14 Acknowledgements 67
15 Normative References 68
16 Informative References 69
17 Author's Address 71
18 Full Copyright Statement 72

4. The SNMP Network Management Framework

The SNMP Management Framework presently consists of five major components:

- An overall architecture, described in RFC 2571 [RFC2571].
- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in RFC 1155 [RFC1155], RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in RFC 2578 [RFC2578], RFC 2579 [RFC2579] and RFC 2580 [RFC2580].
- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901] and RFC 1906 [RFC1906]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [RFC1906], RFC 2572 [RFC2572] and RFC 2574 [RFC2574].
- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [RFC1905].
- A set of fundamental applications described in RFC 2573 [RFC2573] and the view-based access control mechanism described in RFC 2575 [RFC2575].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [RFC2570].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

This memo does not specify a MIB module.

5. Overview

There is a need for a standardized way of defining aggregated data structures for the representation of management information, which can be utilized with existing and future versions of SNMP. The SMIv2 data model is based on groups of rectangular tables, which are related because they share one or more INDEX clause components. This model provides a single containment layer per table, because all the objects in a conceptual row must be simple types (e.g., Integer32, SnmpAdminString, Counter64).

The practice of spreading a multi-layer data structure across several rectangular tables causes MIB modules to be much too verbose, hard to understand, and even harder to implement. The containment relationships between tables are usually described in INDEX clauses and various DESCRIPTION clauses.

This practice has a negative impact on agent implementations, which are harder to implement and test, due to row creation and row activation ordering issues. This practice adds complexity to management application development as well.

Software development and human readability would benefit from a data definition language which more closely represents the basic data structures that exist in almost all programming languages.

[ed. - This revision is intended to introduce the SMI Data Structure concepts and is not yet defined in sufficient detail to be suitable as a formal specification.]

5.1. Terms

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 RFC 2119. [RFC2119]

This document uses some terms that need introduction:

Aggregated Data Object
: This term refers to any data object which provides some sort of containment for other data objects, which is any variable construct other than LEAF (e.g., ARRAY, UNION, or STRUCT).

Data Object
: This term refers to any SMI Data Structure variable declaration, at

any level of containment.

MIB Object
This term generically refers to a SMIv2 OBJECT-TYPE macro definition. It may also refer to an SMI Data Structure definition.

OID This is a shorthand term for 'OBJECT IDENTIFIER'.

LEAF This term refers to any accessible data object with a syntax that resolves to a SMI base type. To avoid confusion, the term appears in capital letters when referring to any data object definition which represents a base type.

SMI Data Structure (SMI-DS)
This term refers to the concepts and definitions defined in this document.

5.2. Design Objectives

The working group objectives for this work are detailed in the SMIng Objectives document [RFC3216]. (Refer to Appendix D for a detailed discussion of each accepted objective.)

The primary high-level design goals of this work are:

- Significantly enhance the usefulness of the SMI as a network management data definition language, by creating a modern programming language like data model supporting aggregated containment.

- Enhance SMI object instance naming to support aggregated hierarchical data structures, while remaining backwardly-compatible with SMIv2 naming.

- Improve readability by enhancing reusability and removing as much redundant text as possible. The SMI should be as easy to use as possible, for the largest number of people. Therefore, a priority hierarchy can be established, starting with MIB readers, then MIB writers, management software developers, and MIB compiler writers.

- Maintain 100% forward and backward translation compatibility with SMIv2. It must be possible to convert all valid SMIv2 constructs to SMI-DS constructs without loss of semantics (i.e., forward compatibility). It should also be possible to translate any SMI-DS construct to one or more SMIv2 constructs, if the associated

feature(s) exist in SMIv2. Refer to Appendix A for details on SMIv2 <--> SMI-DS translations.

- Preserve as many of the SMIv2 mechanisms and 'installed knowledge-base' as possible. There will a transition period lasting several years, in which SMIv2 MIBs will be converted to SMIv3 format. It is important that MIB readers and writers be able to understand both SMI syntaxes during this period, and so it will be beneficial to keep them as close as possible. Clauses that have not changed at all in semantics between SMI versions should maintain the same syntax.

- Make sure accessible data objects (i.e., LEAF objects) can be used with existing versions of SNMP.

There are some relevant topics which not design objectives addressed by this draft:

- Compatibility with any version of ASN.1.

- Equally weighted importance for support of COPS-PR and SNMP. There is a huge disparity in deployment of applications utilizing these protocols. The solution space is biased in favor of SNMP because that will benefit the largest number of people.

- Idiot proof MIB design. Data structures can help better organize the information found in a MIB, but they cannot prevent bad design choices or badly written DESCRIPTION clauses.

5.3. Data Structure Constructs

There are four basic constructs available in the SMI-DS language for the definition of data objects.

LEAF This construct is conceptually equivalent to an OBJECT-TYPE macro definition for an accessible MIB object in SMIv2, except a LEAF can be defined at any level of containment. A LEAF type definition or variable declaration resolves to any SMIng base type. In SMI-DS, all other constructs must eventually resolve to some number of these objects, and only LEAF data objects are actually accessible via SNMP.

ARRAY
This construct provides a multi-dimensional array structure, similar to the SEQUENCE construct in SMIv2. However, instead of

one flat 'row' consisting of only accessible base-type MIB objects, an ARRAY can consist of an arbitrary mix of any of the four types of data object constructs. Only base type data objects can be used in an ARRAY INDEX clause (the same ones as in SMIv2), and the rules for encoding INDEX clause base types in OIDs are the same as for SMIv2.

UNION

This construct provides a mechanism to conceptually allow a single object definition to contain one of potentially several different construct definitions. Only one of these constructs is actually instantiated at any time by the agent. Unlike a union in the C language, the unused union members cannot be accessed at all (no 'cast' operator in SMI).

STRUCT

This construct provides a mechanism to group an arbitrary number of data constructs (of any type), allowing a theoretically unlimited number of data containment layers. It is similar to the ARRAY construct, except there is no INDEX clause.

5.4. Relationship to SMIv2

Whenever possible, existing SMIv2 macros or clauses have been used without modification. Two exceptions are the TEXTUAL-CONVENTION and OBJECT-TYPE macros. In order to reinforce and support a data model more aligned with popular programming concepts and practices, these macros have been replaced by the TYPEDEF and VAR macros (respectively). Strong emphasis is placed on the separation of potentially reusable type definitions and variable declarations. The ASN.1 tabular data model is replaced with a 'hierarchical containment' data model, which is more similar to the 'native' data representation used by the managed device.

The type of declarations that can be made in an SMI-DS module do not really change at all, but some constructs have changed. The major differences between an SMIv2 construct and the equivalent SMI-DS construct are listed in the table below:

SMIv2	SMI-DS
TEXTUAL-CONVENTION	TYPEDEF LEAF
scalar OBJECT-TYPE	VAR LEAF
tabular OBJECT-TYPE	VAR ARRAY
NOTIFICATION-TYPE	NOTIFICATION

Notification semantics have not changed at all, although the syntax has changed slightly to make them more consistent with the TYPEDEF and VAR macros. The ASN.1 specific SEQUENCE macro, and the 'FooTable' and 'FooEntry' OBJECT-TYPE definitions that start every SMIv2 table are removed. The basic SYNTAX clause has not changed at all, except that a new variant is provided to specify a typed OID pointer (see section 5.8).

Many constructs do not change at all, such as the IMPORTS, MODULE-IDENTITY, MAX-ACCESS, STATUS, DESCRIPTION, REFERENCE, DEFVAL, OBJECTS, and MODULE-COMPLIANCE macros.

5.5. Hierarchical Instance Naming

In order to fully utilize the capabilities of arbitrary containment, a new way of naming object instances is needed, which is designed for hierarchical data structures instead of tables, without changing the OID values for any existing SMIv2 objects which are converted to the SMI-DS object naming format.

Since it is possible for accessible objects to exist in the same containment structure as non-accessible objects, it is not possible to name SMI-DS objects with a 'flat' model. SMIv2 assumes all accessible objects in the same containment structure have the same number of object identifier components, and the exact same format for all instance identifier components. This assumption cannot be made for SMI-DS object naming.

This new naming scheme can help reduce implementation complexity for agent and application developers for SNMP Set operations. Currently, associated attributes can be spread across multiple tables, (possibly sharing major indexes) each with their own RowStatus and set of 'SNMP callback' functions. This design approach can get relatively complicated, especially if 'createAndWait' and 'notInService' RowStatus values are supported. By allowing aggregated containment instead of unfolding data structures into tables, implementation of high-level Set operations can be simplified for both agent and application developers.

The basic format of an OID for an SMI-DS data object is not changed from SMIv2. OIDs are constructed left to right. The left fragment contains static OID values which indicate the name of a node in the MIB tree. The right fragment contains potentially dynamic OID values which represent the instance identifier for the node specified by the left fragment.

LEAF Data Object Naming

A SCALAR variable declaration is named as follows:

<oidBase>.0

where:

<oidBase> is a well-formed OID base fragment.

Aggregate Data Object Naming

An Aggregated Data Object variable declaration is named as follows:

<oidBase>.<compatNode>.<childNode>
[.<childNode> ...] [.<indexNode> ...]

where:

<oidBase> is a well-formed OID base fragment, (also called the left anchor).

<compatNode> contains the value 1.

<childNode> is the data object child node identifier, which must be an INTEGER between 1 and 4294967295. (Similar to a column identifier in an SMIv2 table.)

<indexNode> is present only if the variable declaration resolves to a type that contains any ARRAY constructs, and MUST be an INTEGER between 0 and 4294967295. (Similar to an instance identifier in an SMIv2 table.)

SMI-DS OID Construction

OIDs are constructed in an iterative manner, using two conceptual buffers:

base buffer
used for building the static portion of an OID, left to right.

This buffer contains the <oidBase>, <compatNode>, and all <childNode> identifiers.

index buffer
used for building a sequence of ARRAY indexes, (left to right), similar to the instance identifier portion of an SMIv2 OID for a tabular object. This buffer contains all the <indexNode> identifiers.

The expansion algorithm for <childNode> is repeated if it represents an aggregated data object. If it represents an ARRAY construct, then all <indexNode> components for this array type are appended to index buffer.

The algorithm terminates when a LEAF data object is encountered. The index buffer is then appended to the base buffer, to form the complete instance identifier for a specific variable declaration.

5.6. SMI-DS Data Object Usage Examples

The following sections introduce some examples of simple data structures that are currently achieved with relatively verbose text in TEXTUAL-CONVENTION and OBJECT-TYPE DESCRIPTION clauses using SMIv2. Refer to Appendix B for an example of a (somewhat) complete SMI-DS module.

5.6.1. InetAddress Example

The Internet Address textual conventions defined in the "Textual Conventions for Internet Network Addresses" MIB module [RFC2851] defines several variants of an Internet address (InetAddress), and a control object (InetAddressType) to distinguish which variant is actually present in an InetAddress object instance. This construct may be more concisely and properly represented in SMI-DS by a structure containing the control object and a union of all the address variants.

```
-- a union of all the InetAddress types

TYPEDEF UNION InetAddressUnion {
    DESCRIPTION
       "Internet address in 4 different representations."

    LEAF ipUnknown {
       SYNTAX      OCTET STRING (SIZE (0..65535))
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
```

```
            "Represents an Internet address using an externally
             defined format. The associated InetAddressType
             object value is 'unknown(0)'."
    } ::= 1

    LEAF ipv4Addr {
       SYNTAX      InetAddressIPv4
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "Represents an IPv4 Internet address. The
            associated InetAddressType object value
            is 'ipv4(1)'."
    } ::= 2

    LEAF ipv6Addr {
       SYNTAX      InetAddressIPv6
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "Represents an IPv6 Internet address. The
            associated InetAddressType object value
            is 'ipv6(2)'."
    } ::= 3

    LEAF ipDnsAddr {
       SYNTAX      InetAddressDNS
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "Represents an DNS domain name.  The associated
            InetAddressType object value is 'dns(16)'."
    } ::= 4
}

TYPEDEF STRUCT HostInetAddress {
    DESCRIPTION
       "Internet address for an end-station host, adhering
        to the SMIv2 'associated objects' design approach."

    LEAF addrType {
       SYNTAX      InetAddressType
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
```

```
            "The type of Internet address."
    } ::= 1

    UNION addr {
       SYNTAX      InetAddressUnion
       STATUS      current
       DESCRIPTION
           "The Internet address."
    } ::= 2
}

VAR STRUCT myAddress {
    SYNTAX      HostInetAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Internet address of this host."
} ::= { someBase 1 }

VAR UNION newAddress {
    SYNTAX      InetAddressUnion
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Example of the new way to represent a union variable,
         without the use of an associated InetAddressType object."
} ::= { someBase 2 }

Note 1) The accessible object instances defined within this structure
(addrType, ipUnknown, ipv4Addr, ipv6Addr, etc.)  have different lengths:

  myAddress                ::= { someBase 1 }
  myAddress.addrType       ::= { myAddress 1 1 }
  myAddress.addr           ::= { myAddress 1 2 }
  myAddress.addr.ipUnknown ::= { myAddress 1 2 1 }
  myAddress.addr.ipv4Addr  ::= { myAddress 1 2 2 }
  myAddress.addr.ipv6Addr  ::= { myAddress 1 2 3 }
  myAddress.addr.dnsAddr   ::= { myAddress 1 2 4 }

  newAddress               ::= { someBase 2 }
  newAddress.ipUnknown     ::= { newAddress 1 1 }
  newAddress.ipv4Addr      ::= { newAddress 1 2 }
  newAddress.ipv6Addr      ::= { newAddress 1 3 }
  newAddress.dnsAddr       ::= { newAddress 1 4 }
```

Note 2) The mandatory MAX-ACCESS clause within a LEAF construct in a TYPEDEF macro is used to specify the maximum access level that is possible via a management protocol. The optional MAX-ACCESS clause within a VAR macro is used to specify the constrained maximum access level for that specific variable declaration, and must not specify a higher access than declared within a TYPEDEF macro. (E.g., myAddress is a read-only variable even though the LEAF nodes in the HostInetAddress TYPEDEF are read-create. The same LEAF nodes used within the newAddress variable declaration are read-write.) If an overall MAX-ACCESS clause is not present in the VAR macro, then the values specified in the LEAF nodes are used.

Note 3) The addrType field is not actually needed for simple variable declarations, because UNION constructs are instantiated with at most one accessible member. In the example above, a GetNext Request for 'myAddress.addr' or 'newAddress' will return only one type of InetAddress string from the InetAddressUnion. The associated InetAddressType variable is needed only when used together with the InetAddress (generic string form) as INDEX components in an ARRAY.

Note 4) Just like a TEXTUAL-CONVENTION in SMIv2, a TYPEDEF has no instances associated with it and therefore no MIB root assigned. It is only when a a variable of a particular type is declared (and therefore assigned a MIB root) that the full OID for a data object is known.

5.6.2. Generic High Capacity Counter Example

There are many MIBs that contain up to the three OBJECT-TYPE macro definitions for every high capacity counter, in order to accommodate SNMPv1 implementations without support for Counter64 and 32-bit implementations without any high capacity support at all.

A type definition (GenericCounter) for a union that contains an object for each of the three scenarios would better represent the intended semantics of this design, and use less text within data structure definitions than an SMIv2 version. Note that a discriminator object is not needed for a union, because the agent (or management application) will instantiate at most one of the variants.

```
TYPEDEF UNION GenericCounter {
    DESCRIPTION
       "Generic counter for all versions of SNMP."

    LEAF c32 {
       SYNTAX      Counter32
```

```
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The Counter32 representation of the counter."
    } ::= 1

    LEAF c64 {
       SYNTAX      Counter64
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The Counter64 representation of the counter."
    } ::= 2

    STRUCT c32pair {
        DESCRIPTION
            "Pair of Counter32 objects to represent a 64-bit
             counter."

        LEAF c32low {
            SYNTAX      Counter32
            MAX-ACCESS  read-only
            STATUS      deprecated
            DESCRIPTION
                "The lower 32 bits of a 64 bit counter."
        } ::= 1

        LEAF c32hi {
            SYNTAX      Counter32
            MAX-ACCESS  read-only
            STATUS      deprecated
            DESCRIPTION
                "The upper 32 bits of a 64 bit counter."
        } ::= 2
    } ::= 3
}

VAR UNION myCounter {
    SYNTAX      GenericCounter
    STATUS      current
    DESCRIPTION
       "An example generic counter variable."
} ::= { someBase 3 }
```

Note 1) Inline vs. external type definition: The 'c32pair' STRUCT could

have been defined as a separate type and a STRUCT declared with a SYNTAX clause that referenced that type (e.g., <struct-ref-type-decl> form of the STRUCT declaration). The instance numbering works out the same either way.

The following OIDs would be possible for the 'myCounter' variable declaration:

```
myCounter                ::= { someBase 3 }
myCounter.c32            ::= { myCounter 1 1 }
myCounter.c64            ::= { myCounter 1 2 }
myCounter.c32pair        ::= { myCounter 1 3 }
myCounter.c32pair.c32low ::= { myCounter 1 3 1 }
myCounter.c32pair.c32hi  ::= { myCounter 1 3 2 }
```

Note 2) Even though only one node of a UNION can be instantiated at any given time, a GetNext Request for a UNION which contains other aggregated data objects can cause multiple instances to be returned from that sub-tree, as with the 'c32low' and 'c32hi' LEAF objects in the example above.

Note 3) Only the STATUS clauses for LEAF data object definitions are relevant for compliance section usage. However, the above example raises issues regarding an aggregated data object which contains a mixture of current, deprecated, and obsolete LEAF objects. (Is the STATUS of the GenericCounter UNION itself current or deprecated?)

5.6.3. Converted SMIv2 TABLE Example

The following example shows how two objects from the ifTable [RFC2863] would be defined in SMI-DS syntax. Note that in in this example, the interface table is modeled directly as a variable declaration, without using a TYPEDEF. This practice is discouraged for new MIB definitions.

```
-- this is modeled as an ARRAY variable, rather than
-- an ARRAY containing a TYPEDEF'ed structure, to preserve
-- compatibility with SMIv2

VAR ARRAY ifTable {

    DESCRIPTION
        "A list of interface entries.  The number of entries
         is given by the value of ifNumber."

    INDEX { ifIndex }
```

```
    LEAF ifIndex {
        SYNTAX  InterfaceIndex
        MAX-ACCESS  read-only
        STATUS current
        DESCRIPTION
            "A unique value, greater than zero, for each
             interface.  It is recommended that values are assigned
             contiguously starting from 1.  The value for each
             interface sub-layer must remain constant at least from
             one re-initialization of the entity's network
             management system to the next re-initialization."
    } ::= 1

    LEAF ifDescr {
        SYNTAX      DisplayString (SIZE (0..255))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "A textual string containing information about the
             interface.  This string should include the name of the
             manufacturer, the product name and the version of the
             interface hardware/software."
    } ::= 2

    LEAF ifType {
        SYNTAX      IANAifType
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The type of interface.  Additional values for ifType
             are assigned by the Internet Assigned Numbers
             Authority (IANA), through updating the syntax of the
             IANAifType textual convention."
    } ::= 3

    -- rest of ifTable LEAF objects would follow
} ::= { interfaces 2 }

-- declare the ifEntry descriptor for use in other AUGMENTS
ifEntry OBJECT IDENTIFIER ::= { ifTable 1 }
```

Note 1) The object naming and semantics are identical to the SMIv2 version. The OIDs for instance number '17' are shown:

```
  ifTable                  ::= { interfaces 2 }
```

```
  ifTable[17]              ::= Not Available
  ifTable[17].ifIndex      ::= { ifTable 1 1 17 }
  ifTable[17].ifDescr      ::= { ifTable 1 2 17 }
  ifTable[17].ifType       ::= { ifTable 1 3 17 }
```

5.7. Data Structure Augmentations

SMIv2 allows for MIB tables to be conceptually extended over time, without modifying the original MIB table definition, using the AUGMENTS clause. This is usually done to allow vendor extensions to standard MIBs, or to avoid editing a 'stable' RFC.

In SMI-DS, the AUGMENTS clause is preserved and adapted for use with aggregated data objects, in order to maintain backward compatibility with SMIv2. Only inline variable declarations for ARRAY data objects can be augmented.

In addition to the AUGMENTS clause, which models 1:1 existence relationships between two ARRAY variables, a SPARSE-AUGMENTS clause is provides to model conditional 1:1 existence relationships between the augmenting ARRAY variable and the augmented ARRAY variable.

The AUGMENTS construct defines one or more nodes which are conceptually added to the outermost containment layer of the augmented ARRAY variable. The augmenting ARRAY variable inherits all of the index components of that ARRAY (exactly as with SMIv2).

A variant of the AUGMENTS construct is provided (called SPARSE-AUGMENTS) for situations in which a static subset of an existing ARRAY is augmented. The DESCRIPTION clause for an ARRAY which is a sparse augmentation MUST explain the relationship between the augmenting and augmented table.

The AUGMENTS clause in SMIv2 references the internal table node (e.g., ifEntry, not ifTable), but SMI-DS ARRAY variables do not need or use this internal construct. To remain compatible with SMIv2, an OBJECT IDENTIFIER macro is used to declare an object descriptor which can be used in AUGMENTS and SPARSE-AUGMENTS clauses.

AUGMENTS Example

The following trivial example shows how some high-capacity counters and time-related attributes might be added to an existing array of packet

statistics.

```
TYPEDEF ARRAY InetHostStats {
    DESCRIPTION
        "Example of a IP host stats table."

    INDEX { ifIndex, inetAddrType, inetAddr }

    LEAF inetAddrType {
        SYNTAX  InetAddressType
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The IP address type for the array entry.
             The InetAddressType values 'unknown(1)' and
             'dns(16)' are not allowed."
    } ::= 1

    LEAF inetAddr {
        SYNTAX  InetAddress
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The IP address for the array entry."
    } ::= 2

    LEAF inPkts {
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
          "The number of packets received by the specified host
           on the specified interface."
    } ::= 3

    LEAF outPkts {
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
          "The number of packets transmitted by the specified
           host on the specified interface."
    } ::= 4

    -- Octet counters removed to make example shorter
```

```
}

-- variable declaration for a InetHostStats data collection

VAR ARRAY ipStats {
    SYNTAX      InetHostStats
    STATUS      current
    DESCRIPTION
       "The IP host statistics for this network device."
} ::= { someBase 4 }

-- OID declaration to keep AUGMENTS clause consistent
ipStatsEntry OBJECT IDENTIFIER ::= { ipStats 1 }

-- a struct containing additional information for each
-- set of counters

TYPEDEF STRUCT HostStatsTimeData {
    DESCRIPTION
        "Add some times related objects associated with
         each set of counters."

    LEAF createTime {
       SYNTAX      TimeStamp
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
          "The value of sysUpTime at the time this set of
           counters was created."
    } ::= 1

    LEAF updateInterval {
       SYNTAX      Unsigned32
       UNITS       "milliseconds"
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
          "The average amount of time that elapses between
           internal polling intervals for this counter set.
           A value of zero indicates that the counter set
           values are not polled internally."
    } ::= 2
}

-- Augment the ipStats variable with the ipXStats variable:
```

```
--    - 2 HC packet counters
--    - a HostStatsTimeData STRUCT
--    - an ARRAY of InetPortNumber packet counters

VAR ARRAY ipXStats {
    DESCRIPTION
        "Adds HC counters and additional information to
         the ipStats statistics."

    AUGMENTS { ipStatsEntry }

    LEAF inHCPkts {
        SYNTAX      Counter64
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The number of packets received by the specified
             host on the specified interface."
    } ::= 1

    LEAF outHCPkts {
        SYNTAX      Counter64
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The number of packets transmitted by the specified
             host on the specified interface."
    } ::= 2

    -- Octet counters removed to make example shorter

    STRUCT timeData {
        SYNTAX      HostStatsTimeData
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "Additional time-related information."
    } ::= 3

    ARRAY portStats {
        DESCRIPTION
            "Extend the ARRAY with InetPort statistics."

        INDEX { inetPort }
```

```
        LEAF inetPort {
            SYNTAX      InetPortNumber
            MAX-ACCESS  not-accessible
            STATUS      current
            DESCRIPTION
                "The Internet port number for the array entry."
        } ::= 1

        UNION uInPkts {
            SYNTAX      GenericCounter
            MAX-ACCESS  read-only
            STATUS      current
            DESCRIPTION
                "The number of packets received by the specified
                 host on the specified port."
        } ::= 2

        UNION uOutPkts {
            SYNTAX      GenericCounter
            MAX-ACCESS  read-only
            STATUS      current
            DESCRIPTION
                "The number of packets transmitted by the specified
                 host on the specified port."
        } ::= 3

        -- Octet counters removed to make example shorter
    } ::= 4
} ::= { someBase 5 }

ipXStatsEntry   OBJECT IDENTIFIER ::= { ipXStats 1 }
```

Note 1) The following example lists the potential OID values for each of the fields in the 'ipStats' and 'ipXStats' variables in the example above.

In this example only the instances for interface 17, InetAddressType 'ipv4(1)', InetAddress '192.168.0.1', and InetPortNumber '80' are shown.

```
   ipStats                      ::=   { someBase 4 }
   ipStats[17]                  ::=   Not Available
   ipStats[17][1]               ::=   Not Available
   ipStats[17][1][192.168.0.1] ::=   Not Available

   ipStats[17][1][192.168.0.1].inPkts ::=
```

```
       { ipStats 1 3 17 1 4 192 168 0 1 }

   ipStats[17][1][192.168.0.1].outPkts ::=
       { ipStats 1 4 17 1 4 192 168 0 1 }

   ipXStats                    ::=   { someBase 5 }
   ipXStats[17][1][192.168.0.1].inHCPkts ::=
       { ipXStats 1 1 17 1 4 192 168 0 1 }

   ipXStats[17][1][192.168.0.1].outHCPkts ::=
       { ipXStats 1 2 17 1 4 192 168 0 1 }

   ipXStats[17][1][192.168.0.1].timeData ::=
       { ipXStats 1 3 17 1 4 192 168 0 1 }    (not-accessible)

   ipXStats[17][1][192.168.0.1].timeData.createTime ::=
       { ipXStats 1 3 1 17 1 4 192 168 0 1 }

   ipXStats[17][1][192.168.0.1].timeData.updateInterval ::=
       { ipXStats 1 3 2 17 1 4 192 168 0 1 }

   ipXStats[17][1][192.168.0.1].portStats ::=
       { ipXStats 1 4 17 1 4 192 168 0 1 }    (not-accessible)

   ipXStats[17][1][192.168.0.1].portStats[80] ::= Not Available

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts ::=
       { ipXStats 1 4 2 17 1 4 192 168 0 1 80 }   (not-accessible)

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts.c32 ::=
       { ipXStats 1 4 2 1 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts.c64 ::=
       { ipXStats 1 4 2 2 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts.c32pair ::=
       { ipXStats 1 4 2 3 17 1 4 192 168 0 1 80 } (not-accessible)

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts.c32pair.c32low ::=
       { ipXStats 1 4 2 3 1 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uInPkts.c32pair.c32hi ::=
       { ipXStats 1 4 2 3 2 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts ::=
```

```
       { ipXStats 1 4 3 17 1 4 192 168 0 1 80 }   (not-accessible)

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts.c32 ::=
       { ipXStats 1 4 3 1 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts.c64 ::=
       { ipXStats 1 4 3 2 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts.c32pair ::=
       { ipXStats 1 4 3 3 17 1 4 192 168 0 1 80 } (not-accessible)

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts.c32pair.c32low ::=
       { ipXStats 1 4 3 3 1 17 1 4 192 168 0 1 80 }

   ipXStats[17][1][192.168.0.1].portStats[80].uOutPkts.c32pair.c32hi ::=
       { ipXStats 1 4 3 3 2 17 1 4 192 168 0 1 80 }
```

Note 2) Although arbitrary levels of nested containment are theoretically possible, SNMP varbind size limitations and common sense design practices set practical limits on the complexity of data object definitions.

Note 3) The SPPI provides an EXTENDS mechanism, which allows new LEAF objects to be defined in a table which conceptually adds INDEX components to an existing table. This mechanism is accomplished by defining an additional ARRAY (with the new INDEX components and objects) in an AUGMENTS clause, like the 'portStats' example above.

SPARSE-AUGMENTS Example

The following example shows how information about physical sensors may sparsely augment the entPhysicalTable [RFC2737].

```
VAR ARRAY entSensorData {
    DESCRIPTION
        "Adds the ability to read physical sensor values
         to the Entity MIB. An entSensorData object exists
         for each entPhysicalEntry for which the entPhysicalClass
         object value is 'sensor(8)'."
    REFERENCE
        "RFC 2737, section 3."

    SPARSE-AUGMENTS { entPhysicalEntry }
```

```
    LEAF entSensorType {
        SYNTAX        EntitySensorDataType
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The type of data returned by the associated
             entSensorValue object. ..."
    } ::= 1

    LEAF entSensorScale {
        SYNTAX        EntitySensorDataScale
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The exponent to apply to values returned by the
              associated entSensorValue object. ..."
    } ::= 2

    -- rest of entSensorEntry objects would follow ...

} ::= { someBase 6 }
```

Note 1) SMI-DS objects can augment SMIv2 tables, since the SMIv2 <--> SMI-DS conversion algorithms are transparent. The augmented variable object descriptor may be any value that would be accepted in an SMIv2 AUGMENTS clause.

Note 2) The following OIDs would be possible for the 'entSensorEntry' augmentation. The instances for entPhysicalIndex == 17 are shown in this example:

```
   entSensorData                    ::= { someBase 6 }
   entSensorData[17]                ::= Not Available
   entSensorData[17].entSensorType  ::= { entSensorData 1 1 17 }
   entSensorData[17].entSensorScale ::= { entSensorData 1 2 17 }
```

5.8. SYNTAX POINTER Clause

The 'VariablePointer' and 'RowPointer' TEXTUAL-CONVENTIONs [RFC2579] provide semantic constraints on the generic OBJECT IDENTIFIER, but they can only be used to point to a variable or row of any type, not a specific type.

SMI-DS provides a modified SYNTAX clause for object declarations, in

order to specify an OID that must reference a MIB object (LEAF or aggregated data object) of a particular type. The value { 0 0 } is also allowed and is reserved to indicate a NULL pointer.

The form "SYNTAX POINTER <type-name>" specifies an OID which should contain only those values that de-reference to the same type as defined by <type-name>, or contain the NULL pointer value { 0 0 }.

For example, if the RMON DataSource TC [RFC2021] was written in SMI-DS, the POINTER construct might be used as follows:

```
TYPEDEF LEAF DataSource {
    SYNTAX POINTER InterfaceIndex
    MAX-ACCESS     read-create
    STATUS         current
    DESCRIPTION
        "Identifies the source of the data that the associated
         function is configured to analyze. This source can be any
         interface on this device. ...
         For example, if an entry were to receive data from
         interface #1, this object would be set to ifIndex.1."
}
```

Refer to section 6.2 for details on the 'SYNTAX POINTER' clause.

6. Definitions

The follow sections specify the SMI Data Structures syntax and semantics.

[ed. -- this section is intentionally incomplete, because this revision is meant to introduce the SMI Data Structures concepts, syntax, and examples. Complete specification to the level of SMIv2 is TBD.]

6.1. Namespaces

The type names and variable names used in SMI Data Structures are contained is the same namespace, identical to the SMIv2 namespace for OBJECT-TYPE descriptors, and shared with SMIv2. Reserved keywords in SMI-DS or SMIv2 MUST NOT be used as type names or object descriptors.

Ideally, every data object containment level would define its own namespace, in a truly hierarchical fashion. However, this would not be compatible with existing SMIv2 practices, and would require changes to the IMPORTS, MODULE-COMPLIANCE and OBJECT-GROUP macros to support.

[ed. - further definition of namespaces TBD]

6.2. Syntax

[ed. - the following ad-hoc syntax definition is a first-pass attempt, and obviously needs ABNF definition, and a detailed mappings and rules section for each construct. At this time, any construct which is equivalent to the SMIv2 version is not fully specified.]

```
-- top level construction

<module> ::=

    "MODULE" <module-name> "DEFINITIONS" "::=" "BEGIN"
        <imports-decl>
        <module-identity-decl>
        [<module-decl ...]
        [<compliance-section>]
    "END"

<module-name>  ::=  (same as SMIv2)

<imports-decl> ::=  (same as SMIv2)
```

```
<module-identity-decl> ::=  (same as SMIv2)

<module-decl> ::=

     ( <object-identifier> | <object-identity> |
       <typedef-decl> | <var-decl> | notification-decl> )

<object-identifier> ::= (SMIv2 OBJECT IDENTIFIER clause)

<object-identity> ::= (SMIv2 OBJECT-IDENTITY clause)

<typedef-decl> ::=

    "TYPEDEF" ( <leaf-typedef> | <array-typedef> |
                <union-typedef> | <struct-typedef> )

<var-decl> ::=

    "VAR" ( <leaf-var-decl> | <array-var-decl> |
            <union-var-decl>  | <struct-var-decl> )

<leaf-typedef> ::=

    "LEAF" <type-name> <leaf-core-decl>

<type-name> ::=

    (same rules as for SMIv2 TEXTUAL-CONVENTION descriptors)

<leaf-core-decl> ::=

    "{"
        [<display-part>]
        <syntax-clause>
        [<units-clause>]
        <max-access-clause>
        <status-clause>
        <description-clause>
        [<reference-clause>]
        [<defval-clause>]
    "}"

<display-part> ::=   (same as SMIv2 DIPLAY-HINT)

<syntax-clause> ::=
```

```
   ( <plain-syntax-clause> | <pointer-syntax-clause> )

<plain-syntax-clause> ::=

   (same as SMIv2, plus 64-bit numbers and float data types)

<pointer-syntax-clause> ::=

   "SYNTAX" "POINTER" <type-name>

<units-clause> ::= (same as SMIv2)

<max-access-clause> ::= (same as SMIv2)

<status-clause> ::= (same as SMIv2)

<description-clause> ::= (same as SMIv2)

<reference-clause> ::= (same as SMIv2)

<defval-clause> ::= (same as SMIv2)

<leaf-type-decl> ::=

    "LEAF" <object-descriptor> <leaf-core-decl>
        "::=" <N>

<object-descriptor> ::=

    (same rules as for SMIv2 OBJECT-TYPE descriptors)

<N> ::= an INTEGER in the range (1..4294967295)

<leaf-var-decl> ::=

    "LEAF" <object-descriptor> <leaf-core-decl>
        "::=" <oid-assignment>

<oid-assignment> ::=  (same as SMIv2)

<array-typedef> ::=

    "ARRAY" <type-name> "{"
       <description-clause>
       [<reference-clause>]
```

```
       <index-decl>
       <object-decl> [<object-decl> ...]
    "}"

<index-clause> ::=

     ( <index-decl> | <augments-decl> |
       <sparse-augments-decl> )

<index-decl> ::=

    "INDEX" "{" <object-descriptor>
                [ "," <object-descriptor> ...] "}"

<augments-decl> ::=

    "AUGMENTS" "{" <object-descriptor> "}"

<sparse-augments-decl> ::=

    "SPARSE-AUGMENTS" "{" <object-descriptor> "}"

<object-decl> ::=

    ( <leaf-type-decl> | <array-type-decl> |
      <union-type-decl>  | <struct-type-decl> )

<array-type-decl> ::=

    ( <array-inline-type-decl> | <array-ref-type-decl> )

<array-inline-type-decl> ::=

    <array-inline-core-decl> <N>

<array-inline-core-decl> ::=

    "ARRAY" <object-descriptor> "{"
       <description-clause>
       [<reference-clause>]
       <index-decl>
       <object-decl> [<object-decl> ...]
    "}" "::="

<array-ref-type-decl> ::=
```

```
    <array-ref-core-decl> <N>

<array-ref-core-decl> ::=

    "ARRAY" <object-descriptor> "{"
       <syntax-clause>
       [<max-access-clause>]
       <status-clause>
       <description-clause>
       [<reference-clause>]
    "}" "::="

<array-var-decl> ::=

   ( <array-inline-var-decl> | <array-ref-var-decl> )

<array-inline-var-decl> ::=

    <array-inline-core-decl> <oid-assignment>

<array-ref-var-decl> ::=

    <array-ref-core-decl> <oid-assignment>

<union-typedef> ::=

    "UNION" <type-name> "{"
        <description-clause>
        [<reference-clause>]
        <object-decl> [<object-decl> ...]
    "}"

<union-type-decl> ::=

    ( <union-inline-type-decl> | <union-ref-type-decl> )

<union-inline-type-decl> ::=

    <union-inline-core-decl> <N>

<union-inline-core-decl> ::=

    "UNION" <object-descriptor> "{"
        <description-clause>
        [<reference-clause>]
```

```
        <object-decl> [<object-decl> ...]
    "}" "::="

<union-ref-type-decl> ::=

    <union-ref-core-decl> <N>

<union-ref-core-decl> ::=

    "UNION" <object-descriptor> "{"
        <syntax-clause>
        [<max-access-clause>]
        <status-clause>
        <description-clause>
        [<reference-clause>]
    "}" "::="

<union-var-decl> ::=

    ( <union-inline-var-decl> | <union-ref-var-decl> )

<union-inline-var-decl> ::=

    <union-inline-core-decl> <oid-assignment>

<union-ref-var-decl> ::=

    <union-ref-core-decl> <oid-assignment>

<struct-typedef> ::=

    "STRUCT" <type-name> "{"
        <description-clause>
        [<reference-clause>]
        <object-decl>  [<object-decl> ...]
    "}"

<struct-type-decl> ::=

    ( <struct-inline-type-decl> | <struct-ref-type-decl> )

<struct-inline-type-decl> ::=

    <struct-inline-core-decl> <N>
```

```
<struct-inline-core-decl> ::=

    "STRUCT" <object-descriptor> "{"
        <description-clause>
        [<reference-clause>]
        <object-decl>  [<object-decl> ...]
    "}" "::="

<struct-ref-type-decl> ::=

    <struct-ref-core-decl> <N>

<struct-ref-core-decl> ::=

    "STRUCT" <object-descriptor> "{"
        <syntax-clause>
        [<max-access-clause>]
        <status-clause>
        <description-clause>
        [<reference-clause>]
    "}" "::="

<struct-var-decl> ::=

    ( <struct-inline-var-decl> | <struct-ref-var-decl> )

<struct-inline-var-decl> ::=

    <struct-inline-core-decl> <oid-assignment>

<struct-ref-var-decl> ::=

    <struct-ref-core-decl> <oid-assignment>

<notification-decl> ::=

    "NOTIFICATION" <object-descriptor> "{"
        [<objects-part>]
        <status-clause>
        <description-clause>
        [<reference-clause>]
    "}" "::=" <oid-assignment>

<objects-part> ::=
```

```
    "OBJECTS" "{" <object-descriptor>
                  [ "," <object-descriptor> ...] "}"

<compliance-section> ::=

    (same as SMIv2, except VAR node descriptors need to
     be fully qualified)


-- END
```

7. Information Modules

TBD - This section (and 7 more) need to be completed by adapting sections 3 - 10 of SMIv2 [RFC2578].

8. Appendix A: SMIv2 Compatibility

It is important to advance SMI features in a way that maximizes the reusability of existing SMIv2-based development work and training. Several SMI-DS features are intended to provide mechanisms for automatic (or semi-automatic) translations between SMIv2 and SMI-DS definitions.

8.1. Common Constructs

The following macros, clauses, and keywords are identical in SMIv2 and SMI-DS, and therefore no translation is required. Clauses listed here are not mentioned in the sections describing macro conversions that utilize these clauses.

- BEGIN
- DEFVAL
- DEFINITIONS
- DESCRIPTION
- DISPLAY-HINT
- END
- IMPORTS
- INDEX
- MAX-ACCESS
- MODULE-COMPLIANCE (all clauses)
- MODULE-IDENTITY (all clauses)
- OBJECT-IDENTITY
- OBJECT-IDENTIFIER
- OBJECTS
- REFERENCE
- STATUS
- UNITS

8.2. SMIv2 to SMI-DS Module Conversion

The following SMIv2 macros, clauses and keywords require some conversion:

- NOTIFICATION-TYPE
- OBJECT-TYPE
- SEQUENCE
- TEXTUAL-CONVENTION

TEXTUAL-CONVENTIONs

The TEXTUAL-CONVENTION macro is replaced by the TYPEDEF macro, which can be used to define aggregated data types, in addition to the refinement of base types. The TEXTUAL-CONVENTION macro is replaced with the TYPEDEF macro as follows:

a) prefix type name with 'TYPEDEF LEAF ' and append it with ' {'

b) remove '::= TEXTUAL-CONVENTION'

c) The SYNTAX clause can be modified to refine another LEAF TYPEDEF, or an OBJECT IDENTIFIER type can be changed to a typed OID pointer (e.g., 'SYNTAX POINTER FooType')

d) add a MAX-ACCESS clause specifying the maximum access level for the data type, as used in any possible situation

e) a UNITS clause may be added if appropriate

f) a DEFVAL clause may be added if appropriate

g) end TYPEDEF macro with a '}' token

e.g:

```
FooString ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "This data type is used to model an administratively
         controlled textual string."
    SYNTAX OCTET STRING (SIZE (0..127))
```

is changed to:

```
TYPEDEF LEAF FooString {
    SYNTAX      OCTET STRING (SIZE (0..127))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This data type is used to model an administratively
         controlled textual string."
}
```

OBJECT-TYPE Macro

The generic OBJECT-TYPE macro is replaced with the VAR macro.

Scalar Objects

The scalar OBJECT-TYPE macro is replaced with the 'VAR LEAF' macro as follows:

a) prefix scalar name with 'VAR LEAF ' and append it with ' {'

b) remove '::= OBJECT-TYPE'

c) The SYNTAX clause of OBJECT IDENTIFIER can be changed to a typed OID pointer (e.g., 'SYNTAX POINTER FooType')

d) prefix '::= <oid-assignment>' with a '}' token

e.g.,

```
   sysUpTime OBJECT-TYPE
       SYNTAX      TimeTicks
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The time (in hundredths of a second) since the network
           management portion of the system was last re-initialized."
       ::= { system 3 }
```

is replaced with:

```
   VAR LEAF sysUpTime {
       SYNTAX      TimeTicks
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The time (in hundredths of a second) since the network
           management portion of the system was last re-initialized."
   } ::= { system 3 }
```

Tabular Objects

The tabular OBJECT-TYPE macro is replaced with the 'VAR ARRAY' macro as follows:

a) The contents of the SEQUENCE can be converted in three ways:
   1) placed directly in a VAR ARRAY macro
   2) placed in a STRUCT TYPEDEF and a data node of that type
      declared in the VAR ARRAY macro
   3) placed an ARRAY TYPEDEF, including the INDEX, and a
      variable of this type declared with the VAR ARRAY macro.
      This method must be used to convert tables using the
      AUGMENTS clause.

   The direct method (1) is shown here.

b) The OBJECT-TYPE macro for the table itself (e.g., fooTable)
   is transformed into a VAR ARRAY declaration by extracting
   the object descriptor, prefixing it with 'VAR ARRAY ' and
   appending it with ' {'. The DESCRIPTION clause should be
   transferred and modified as needed.

c) The OBJECT-TYPE macro for the table entry (e.g., fooEntry) is
   discarded except for the INDEX clause, and any information
   from the DESCRIPTION clause is transferred and modified as
   needed. An OBJECT IDENTIFIER macro may be created to
   declare the descriptor for the table entry, allowing it
   to be used in an AUGMENTS or SPARSE-AUGMENTS clause in
   another ARRAY variable declaration. E.g.,

```
  fooEntry  OBJECT IDENTIFIER ::= { fooTable 1 }
```

d) For each OBJECT-TYPE macro, an <object-decl>
   for a 'LEAF' is created.
   - prefix object descriptor with 'VAR LEAF ' and append it
     with ' {'
   - remove '::= OBJECT-TYPE'
   - The SYNTAX clause of OBJECT IDENTIFIER may be changed to a
     typed OID pointer (e.g., 'SYNTAX POINTER FooType')
   - replace '::= { fooEntry <N> }' with '} ::= <N>'

e) prefix a '}' token to the node assignment for the table itself
   (e.g., 'fooTable'), which becomes the node assignment for the
   ARRAY variable declaration.

E.g., (Note: IF-MIB [RFC2863] example DESCRIPTION clauses truncated),

```
   ifStackTable  OBJECT-TYPE
       SYNTAX        SEQUENCE OF IfStackEntry
       MAX-ACCESS    not-accessible
```

```
        STATUS       current
        DESCRIPTION
            "The table containing information on the relationships
             between the multiple sub-layers of network interfaces..."
        ::= { ifMIBObjects 2 }

    ifStackEntry  OBJECT-TYPE
        SYNTAX       IfStackEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
            "Information on a particular relationship between two
             sub-layers, specifying that one sub-layer runs on
             'top' of the other sub-layer.  Each sub-layer
             corresponds to a conceptual row in the ifTable."
        INDEX { ifStackHigherLayer, ifStackLowerLayer }
        ::= { ifStackTable 1 }

    IfStackEntry ::=
       SEQUENCE {
           ifStackHigherLayer  Integer32,
           ifStackLowerLayer   Integer32,
           ifStackStatus       RowStatus
        }

   ifStackHigherLayer  OBJECT-TYPE
        SYNTAX       Integer32
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
            "The value of ifIndex corresponding to the higher
             sub-layer of the relationship, i.e., the sub-layer..."
        ::= { ifStackEntry 1 }

   ifStackLowerLayer  OBJECT-TYPE
        SYNTAX       Integer32
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
            "The value of ifIndex corresponding to the lower sub-
             layer of the relationship, i.e., the sub-layer which ..."
        ::= { ifStackEntry 2 }

   ifStackStatus  OBJECT-TYPE
       SYNTAX        RowStatus
```

```
       MAX-ACCESS     read-create
       STATUS         current
       DESCRIPTION
            "The status of the relationship between two sub-
             layers. ..."
       ::= { ifStackEntry 3 }

 is replaced with:

    VAR ARRAY ifStackTable {
        DESCRIPTION
            "The table containing information on the relationships
             between the multiple sub-layers of network interfaces...

             Information on a particular relationship between two
             sub-layers, specifying that one sub-layer runs on
             'top' of the other sub-layer.  Each sub-layer
             corresponds to a conceptual row in the ifTable."

        INDEX { ifStackHigherLayer, ifStackLowerLayer }

        LEAF ifStackHigherLayer {
            SYNTAX        Integer32
            MAX-ACCESS    not-accessible
            STATUS        current
            DESCRIPTION
                "The value of ifIndex corresponding to the
                 higher sub-layer of the relationship, i.e.,
                 the sub-layer..."
        } ::= 1

        LEAF ifStackLowerLayer {
            SYNTAX        Integer32
            MAX-ACCESS    not-accessible
            STATUS        current
            DESCRIPTION
                "The value of ifIndex corresponding to the
                 lower sub-layer of the relationship, i.e.,
                 the sub-layer which ..."
        }  ::= 2

        LEAF ifStackStatus {
            SYNTAX         RowStatus
            MAX-ACCESS     read-create
            STATUS         current
```

```
            DESCRIPTION
                "The status of the relationship between two sub-
                 layers. ..."
        } ::= 3
    ::= { ifMIBObjects 2 }

    OBJECT IDENTIFIER ifStackEntry ::= { ifStackTable 1 }
```

Notifications

The SMIv2 NOTIFICATION-TYPE macro is replaced with the NOTIFICATION macro as follows:

a) prefix notification name with 'NOTIFICATION ' and append it with ' {'

b) remove '::= NOTIFICATION-TYPE'

c) prefix '::= <oid-assignment>' with a '}' token

e.g.,

```
   linkUp NOTIFICATION-TYPE
       OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }
       STATUS  current
       DESCRIPTION
           "A linkDown trap signifies that the SNMPv2 entity,
            acting in an agent role, has detected that the
            ifOperStatus object for one of its communication links
            left the down state and transitioned into some other
            state (but not into the notPresent state).  This other
            state is indicated by the included value of
            ifOperStatus."
      ::= { snmpTraps 4 }
```

is replaced with:

```
   NOTIFICATION linkUp {
       OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }
       STATUS  current
       DESCRIPTION
           "A linkDown trap signifies that the SNMPv2 entity,
            acting in an agent role, has detected that the
            ifOperStatus object for one of its communication links
```

```
            left the down state and transitioned into some other
            state (but not into the notPresent state).  This other
            state is indicated by the included value of
            ifOperStatus."
    } ::= { snmpTraps 4 }
```

8.3. SMI-DS to SMIv2 Module Conversion

Just as with the transition from SMIv1 to SMIv2, not all new constructs can be efficiently mapped backward (from SMI-DS to SMIv2). Since some new clauses are designed to extract information buried in DESCRIPTION clauses or comments, it is to be expected that backward conversion consists of putting this information back where it came from.

[Guidelines for unfolding tables TBD]

8.4. Compatibility Guidelines

The following guidelines are provided to assist MIB writers create SMI-DS modules that can be properly mapped backward into SMIv2 syntax and semantics.

ARRAYs

The IMPLIED keyword SHOULD NOT be used, except to convert an SMIv2 table which has an IMPLIED INDEX component to SMI-DS. Only one IMPLIED keyword can be used, and it MUST be in the innermost ARRAY construct, if nested ARRAYs are defined. The IMPLIED keyword severely limits the ability to reuse a TYPEDEF containing it, and SHOULD NOT be used in type definitions.

9. Appendix B: Complete MODULE Example

The following example shows a somewhat complete MIB module, adapted from the Remote Monitoring Extensions for Differentiated Services document [DSMON-MIB]. Refer to that document to compare the SMIv2 and SMI-DS definitions.

This is not a transparent conversion of the SMIv2 version, but rather an 'upgraded' version, in which the containment features (such as STRUCTs and nested ARRAYs) are utilized. The intent is to demonstrate how a read-create data structure spread over three tables with SMIv2 can be defined as a single structure with SMI-DS.

```
MODULE DSMON-MIB DEFINITIONS ::= BEGIN

-- partial IMPORTS, only for the aggregation control objects

IMPORTS
        MODULE-IDENTITY, Integer32, Counter32
                FROM SNMPv2-SMI
        MODULE-COMPLIANCE, OBJECT-GROUP
                FROM SNMPv2-CONF
        RowStatus, TimeStamp, TruthValue
                FROM SNMPv2-TC
        OwnerString, rmon
                FROM RMON-MIB
        SnmpAdminString
                FROM SNMP-FRAMEWORK-MIB
        Dscp
                FROM DIFFSERV-DSCP-TC;

-- the MODULE-IDENTITY macro is not changed at all

dsmonMIB MODULE-IDENTITY
    LAST-UPDATED    "200111050000Z"
    ORGANIZATION    "IETF RMONMIB Working Group"
    CONTACT-INFO
            "Same as SMIv2"
    DESCRIPTION
            "Same as SMIv2"
    REVISION  "200111050000Z"
       DESCRIPTION
            "Same as SMIv2"
  ::= { rmon 26 }
```

```
dsmonObjects       OBJECT IDENTIFIER ::= { dsmonMIB 1 }
dsmonNotifications OBJECT IDENTIFIER ::= { dsmonMIB 2 }
dsmonConformance   OBJECT IDENTIFIER ::= { dsmonMIB 3 }

dsmonAggObjects    OBJECT IDENTIFIER ::= { dsmonObjects 1 }

-- the following objects removed from the example
dsmonStatsObjects  OBJECT IDENTIFIER ::= { dsmonObjects 2 }
dsmonPdistObjects  OBJECT IDENTIFIER ::= { dsmonObjects 3 }
dsmonHostObjects   OBJECT IDENTIFIER ::= { dsmonObjects 4 }
dsmonCapsObjects   OBJECT IDENTIFIER ::= { dsmonObjects 5 }
dsmonMatrixObjects OBJECT IDENTIFIER ::= { dsmonObjects 6 }

-- converted DsmonCounterAggGroupIndex TC to a TYPEDEF

TYPEDEF LEAF DsmonCounterAggGroupIndex {
    SYNTAX     Integer32 (0..2147483647)
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
       "This TC describes a data type which identifies a DSMON
        counter aggregation group, ..."
}

-- converted DsmonCounterAggProfileIndex TC to a TYPEDEF

TYPEDEF LEAF DsmonCounterAggProfileIndex {
    SYNTAX     Integer32 (1..2147483647)
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
        "This TC describes a data type which identifies a DSMON
         counter aggregation profile, ..."
}

-- converted dsmonAggProfileTable

TYPEDEF ARRAY DsmonCounterAggProfile {
    DESCRIPTION
        "Controls the setup of a single aggregation profile,
         for which every DSCP value MUST be configured
         into exactly one aggregation group. ..."

    INDEX { dsmonAggProfileDSCP }
```

```
    LEAF dsmonAggProfileDSCP {
        SYNTAX     Dscp
        MAX-ACCESS not-accessible
        STATUS     curent
        DESCRIPTION
            "The specific DSCP value which is configured in an
             aggregation group by this entry."
    } ::= 1

    LEAF dsmonAggGroupIndex {
        SYNTAX      DsmonCounterAggGroupIndex
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The aggregation group which contains this DSCP
             value. ..."
        DEFVAL { 0 }
    } ::= 2
}

-- converted dsmonAggGroupTable

TYPEDEF ARRAY DsmonCounterAggGroup {
    DESCRIPTION
        "Controls the setup of a single aggregation profile,
         for which every DSCP value MUST be configured
         into exactly one aggregation group. ..."

    INDEX { dsmonAggGroupIndex }

    LEAF dsmonAggGroupIndex {
        SYNTAX     DsmonCounterAggGroupIndex
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
            "The specific Aggregation Group which is represented
             group by each entry."
    } ::= 1

    LEAF dsmonAggGroupDescr {
        SYNTAX      SnmpAdminString (SIZE(0..64))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "An administratively assigned description of the
```

```
             aggregation group identified by this entry. ..."
    } ::= 2
}

-- converted dsmonAggControlTable

TYPEDEF STRUCT DsmonCounterAggControl {
    DESCRIPTION
        "Provides an overall description and control
      point for a single aggregation control configuration. ..."

    LEAF dsmonAggControlDescr {
        SYNTAX      SnmpAdminString (SIZE(0..64))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "An administratively assigned description of the aggregation
            profile identified by this entry. ..."
    } ::= 1

    ARRAY aggProfile {
        SYNTAX      DsmonCounterAggProfile
        STATUS      current
        DESCRIPTION
           "A set of DSCP to Aggregation Group mappings."
    } ::= 2

    ARRAY aggGroup {
        SYNTAX      DsmonCounterAggGroup
        STATUS      current
        DESCRIPTION
            "A set of Aggregation Group descriptions."
    } ::= 3

    LEAF dsmonAggControlOwner {
        SYNTAX     OwnerString
        MAX-ACCESS read-create
        STATUS     current
        DESCRIPTION
            "The entity that configured this object and is
             therefore using the resources assigned to it."
    } ::= 4

    LEAF dsmonAggControlStatus {
        SYNTAX      RowStatus
```

```
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this entire aggregation control
             object. ..."
    } ::= 5
}

--
-- variable declarations for the 4 scalars in this group
--

VAR LEAF dsmonMaxAggGroups {
    SYNTAX      Integer32 (2..64)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The maximum number of aggregation groups that this agent
         can support. ..."
} ::= { dsmonAggObjects 1 }

VAR LEAF dsmonAggControlLocked {
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Controls the setup of aggregation groups for this agent. ..."
} ::= { dsmonAggObjects 2 }

VAR LEAF dsmonAggControlChanges {
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This object counts the number of times the value of the
         dsmonAggControlLocked object has changed. ..."
} ::= { dsmonAggObjects 3 }

VAR LEAF dsmonAggControlLastChangeTime {
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This object identifies the value of sysUpTime at the moment
         the dsmonAggControlLocked object was last modified. ..."
```

```
} ::= { dsmonAggObjects 4 }

-- finishing the dsmonAggControlTable by allowing multiple
-- instances of an aggregation control block

VAR ARRAY dsmonAggProfiles {
    STATUS      current
    DESCRIPTION
        "A collection of DSMON aggregation control profiles. ..."

    INDEX { dsmonAggControlIndex }

    LEAF dsmonAggControlIndex {
        SYNTAX     DsmonCounterAggProfileIndex
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
            "The specific Counter Aggregation Profile which is
             represented by each entry."
    } ::= 1

    STRUCT aggControl {
        SYNTAX DsmonCounterAggControl
        STATUS current
        DESCRIPTION
           "The DSMON Counter Aggregation Control entry for
            each profile."
    } ::= 2
} ::= { dsmonAggObjects 5 }

-- No NOTIFICATION-TYPE macros defined in this module

-- Compliance section (currently unchanged from SMIv2)

dsmonCounterAggControlCompliance MODULE-COMPLIANCE
    STATUS  current
    DESCRIPTION
            "Example compliance for the aggregation control
             portion of the DSMON-MIB module."
    MODULE  -- this module
        MANDATORY-GROUPS { dsmonCounterAggControlGroup }

    ::= { dsmonCompliances 1 }

dsmonCounterAggControlGroup OBJECT-GROUP
```

```
    OBJECTS {
     dsmonMaxAggGroups,
     dsmonAggControlLocked,
     dsmonAggControlChanges,
     dsmonAggControlLastChangeTime,
     dsmonAggProfiles.aggControl.dsmonAggControlDescr,
     dsmonAggProfiles.aggControl.dsmonAggControlOwner,
     dsmonAggProfiles.aggControl.dsmonAggControlStatus,
     dsmonAggProfiles.aggControl.appProfile.dsmonAggGroupIndex,
     dsmonAggProfiles.aggControl.appGroup.dsmonAggGroupDescr
    }
    STATUS  current
    DESCRIPTION
        "A collection of objects used to configure and manage
        aggregation groups for DSMON collection purposes."
    ::= { dsmonGroups 1 }

END
```

Note 1) The following example shows the difference between SMIv2 naming and SMI-DS naming, for the OBJECT IDENTIFIERS in the DSMON-MIB module example above.

```
 Object Instance Examples
------------------------------
 O=Old (SMIv2), N=New (SMI-DS)

dsmonAggGroup scalars:
   dsmonMaxAggGroups
     O: dsmonAggObjects.1.0
     N: dsmonAggObjects.1.0
   dsmonAggControlLocked
     O: dsmonAggObjects.2.0
     N: dsmonAggObjects.2.0
   dsmonAggControlChanges
     O: dsmonAggObjects.3.0
     N: dsmonAggObjects.3.0
   dsmonAggControlLastChangeTime
     O: dsmonAggObjects.4.0
     N: dsmonAggObjects.4.0

dsmonAggControlTable example for row 77:
   dsmonAggControlDescr
     O: dsmonAggObjects.5.1.2.77
     N: dsmonAggObjects.5.1.2.1.77
```

```
   dsmonAggControlOwner
     O: dsmonAggObjects.5.1.3.77
     N: dsmonAggObjects.5.1.2.4.77
   dsmonAggControlStatus
     O: dsmonAggObjects.5.1.3.77
     N: dsmonAggObjects.5.1.2.5.77

dsmonAggProfileTable example for row 77.22:
   dsmonAggGroupIndex
     O: dsmonAggObjects.6.1.2.77.22
     N: dsmonAggObjects.5.1.2.2.2.77.22

dsmonAggGroupTable example for row 77.44:
   dsmonAggGroupDescr
     O: dsmonAggObjects.7.1.1.77.44
     N: dsmonAggObjects.5.1.2.3.2.77.44
   dsmonAggGroupStatus
     O: dsmonAggObjects.7.1.2.77.44
     N: not needed because dsmonAggControlStatus
        controls an entire dsmonAggControl data object

Note 2) Scalar object naming does not change at all

Note 3) DSMON Counter Aggregation control requires three tables in SMIv2
(dsmonAggObjects.5 - 7) and one in SMI-DS (dsmonAggObjects.5). This
allows the subordinate RowStatus object (dsmonAggGroupStatus) to be
removed. It also allows the agent to identify the complete hierarchical
position of any object instance by inspection. These implementation
benefits (and others) can help significantly to reduce the software
development costs for complex MIBs.

Note 4) Aggregate object descriptors have to be fully qualified, for
each VAR declaration. Need to consider a shorthand notation in next
version of SMI-DS.
```

10. Appendix C: Open Issues

The following open issues (in no particular order) need to be addressed.

1) SPPI Merge

The biggest issue is SPPI OID naming. Experts in COPS-PR and SPPI should determine how SPPI naming, tabular data model, and various SPPI clauses should be integrated into SMI-DS. This should be done in a way

that does not impact the overall complexity or ease of use as an SMIv2 replacement, possibly contained in a separate document.

2) Conformance Granularity

The concept of MIB conformance may need to change to better handle the complexity created by the type definition and containment features of SMI-DS. MODULE-COMPLIANCE macros for complex data objects may need to allow for automatic conformance update mechanisms. The 'copy-by-reference' property of nested data structures needs to somehow translate to the conformance section. E.g., if 'fooObject1' is deprecated and updated with 'fooObject2' in the 'FooStruct', then the update occurs everywhere a 'FooStruct' is nested. The MODULE-COMPLIANCE needs to be updated somehow for every VAR declaration that is, or has an embedded 'FooStruct'.

3) Conformance Instance Overlap

Since descriptors can occur in TYPEDEFs, they are not unique for conformance purposes (as raised by Randy Presuhn in SLC). An efficient MODULE-COMPLIANCE mechanism is needed to provide conformance info for each VAR and NOTIFICATION declaration, not for each accessible object descriptor. This way, object descriptors can have different conformance requirements at the granularity of the VAR macro.

4) SMIv2 Merge Issues

Sections 3 - 10 of RFC 2578 need to be adapted and added into this document. The extensive set of implementation rules and guidelines needs to be updated and clarified. Complete 'ASN.1 free' syntax needs to be finished, along with the SMIv2 compatibility and transformation guidelines.

5) Base Data Type Extensions

The data types defined in the 'SMIng Core Modules' document should be used by this document somehow.

6) SMI Syntax

Although it is tempting to completely change the syntax for the data definition language to benefit potential 'new users', this would increase overall complexity for new and old users of the SMI. There are many more MIB modules now then April 1993, when SMIv2 was first published as RFC 1442. It took years to convert all the standards track

modules from SMIv1 to SMIv2, and it will probably take years to convert them all from SMIv2 to SMIv3. During the transition, operators and developers need to know both syntax variants, and it will help a great deal if they are similar to each other.

7) STATUS clause for aggregate data objects

It may be useful to have a STATUS clause for an entire aggregate TYPEDEF or VAR construct, which overrides the status of any of the individual nodes within that aggregate. This would allow a simpler way to deprecate the entire object when needed.

11. Appendix D: Discussion of SMIng Objectives

This section lists each accepted design objective described in the SMIng Objectives document [SMING_OBJ], and explains how SMI-DS addresses the objective.

4.1.1 The Set of Specification Documents [Yes]

Description
: SMIv2 is defined in three documents, based on an obsolete ITU ASN.1 specification. SPPI is defined in one document, based on SMIv2. The core of SMIng must be defined in one document and must be independent of external specifications.

Fulfillment
: SMI-DS can meet this objective by simply placing as much text as desired in a single document.

4.1.2 Textual Representation [Yes]

Description
: SMIng definitions must be represented in a textual format.

Fulfillment
: SMI-DS meets this objective because it is specified using only textual characters.

4.1.3 Human Readability [Yes]

Description
: The syntax must make it easy for humans to directly read and write SMIng modules. It must be possible for SMIng module authors to produce SMIng modules with text editing tools.

Fulfillment
The SMI-DS syntax is very close (or identical) to SMIv2 in all respects, so it will be easy for MIB authors and readers to use.

4.1.4 Rigorously Defined Syntax [Yes - TBD]

Description
There must be a rigorously defined syntax for the SMIng language.

Fulfillment
Once the features (and the syntax for those features) are finalized, all SMI-DS constructs will be rigorously defined, including the constructs which do not change from SMIv2.

4.1.5 Accessibility [Yes]

Description
Attribute definitions must indicate whether attributes can be read, written, created, deleted, and whether they are accessible for notifications, or are not accessible. Align PIB-ACCESS and MAX-ACCESS, and PIB-MIN-ACCESS and MIN-ACCESS.

Fulfillment
The MAX-ACCESS clause is retained from SMIv2. PIB versions of these constructs do not really differ in semantics, just in name. PIBs and MIBs use the same MAX-ACCESS clause.

4.1.6 Language Extensibility [Maybe]

Description
The language must have characteristics, so that future modules can contain information of future syntax without breaking original SMIng parsers.

Fulfillment
Although this objective benefits very few people, it can be achieved by rigorously defining the SMI-DS syntax so that a parser can always determine where a construct begins and ends.

4.1.7 Special Characters in Text [No]

Description
Allow an escaping mechanism to encode special characters, e.g., double quotes and new-line characters, in text such as DESCRIPTIONs or REFERENCEs.

Fulfillment
Currently there are no mechanisms added to these SMIv2 constructs used without modification in SMI-DS. It is not clear why forcing the author to use single quotes is unreasonable. Not sure why this is a problem. Adding cryptic character sequences conflicts with objective 4.1.3.

4.1.8 Naming [Yes]

Description
SMIng must provide mechanisms to uniquely identify attributes, groups of attributes, and events. It is necessary to specify how name collisions are handled.

Fulfillment
SMI-DS meets all these requirements. Namespaces are handled the same as in SMIv2.

4.1.9 Namespace Control [Yes]

Description
There must be a hierarchical, centrally-controlled namespace for standard named items, and a distributed namespace must be supported to allow vendor-specific naming and to assure unique module names across vendors and organizations.

Fulfillment
SMI-DS meets this requirement by providing true hierarchical naming, which is compatible with SMIv2 objects. Enterprise-specific definitions and augmentations are supported.

4.1.10 Modules [Yes]

Description
SMIng must provide a mechanism for uniquely identifying a module, and specifying the status, contact person, revision information, and the purpose of a module. SMIng must provide mechanisms to group definitions into modules and it must provide rules for referencing definitions from other modules.

Fulfillment
SMI-DS information modules are conceptually identical to SMIv2 information modules, including the IMPORTS clause.

4.1.11 Module Conformance [Yes]

Description
SMIng must provide mechanisms to detail the minimum requirements implementers must meet to claim conformance to a standard based on the module.

Fulfillment
SMI-DS conformance constructs (such as MAX-ACCESS, MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP) are mostly unchanged from SMIv2.

4.1.12 Arbitrary Unambiguous Identities [Yes]

Description
SMI allows the use of OBJECT-IDENTITIES to define unambiguous identities without the need of a central registry. SMI uses OIDs to represent values that represent references to such identities. SMIng needs a similar mechanism (a statement to register identities, and a base type to represent values).

Fulfillment
Base type semantics (including OBJECT IDENTIFIER) are unchanged from SMIv2.

4.1.13 Protocol Independence [Yes - TBD]

Description
SMIng must define data definitions in support of the SNMP and COPS-PR protocols. SMIng may define data definitions in support of other protocols.

Fulfillment
SMI-DS is fully compatible with SMIv2 and the SNMP protocol. Specific mapping algorithms for COPS-PR object naming are TBD.

4.1.14 Protocol Mapping [Yes]

Description
The SMIng working group, in accordance with the working group charter, will define mappings of protocol independent data definitions to protocols based upon installed implementations. The SMIng working group can define mappings to other protocols as long as this does not impede the progress on other objectives.

Fulfillment
As long as the protocol is actually independent of the data definition language and its naming scheme (as advertised with SNMP), accessible data objects (i.e., LEAF objects) can be manipulated in the same manner as accessible SMIv2 objects.

4.1.15 Translation to Other Data Definition Languages [Yes - TBD]

Description
SMIng language constructs must, wherever possible, be translatable to SMIv2 and SPPI. At the time of standardization of a SMIng language, existing SMIv2 MIBs and SPPI PIBs on the standards track will not be required to be translated to the SMIng language. New MIBs/PIBs will be defined using the SMIng language.

Fulfillment
Algorithms can be specified to convey each SMI-DS construct to one or more SMIv2 constructs. Complex nesting must be unfolded into a set of associated SMIv2 tables, each table corresponding to the accessible objects at a given nest level of the SMI-DS object. Existing SMIv2 tables can easily be converted to SMI-DS using the ARRAY construct.

4.1.16 Base Data Types [Yes]

Description
SMIng must support the base data types Integer32, Unsigned32, Integer64, Unsigned64, Enumeration, Bits, OctetString, and OID.

Fulfillment
The SMIv2 base data types are unchanged in SMI-DS. The Integer64 and Unsigned64 base data types will also be added.

4.1.17 Enumerations [Yes]

Description
SMIng must provide support for enumerations. Enumerated values must be a part of the enumeration definition.

Fulfillment
SMI-DS provides enumerated INTEGERs, unchanged from SMIv2.

4.1.18 Discriminated Unions [Yes]

Description
SMIng must support discriminated unions.

Fulfillment
SMI-DS provides the UNION construct to explicitly define (in a manner that can be machine-parsed) a group of objects with the characteristics of a discriminated union. A STRUCT can be defined which includes the discriminator LEAF object and the UNION object, to further express these semantics. (See HostInetAddress example in section 5.6.1).

4.1.19 Instance Pointers [Yes]

Description
SMIng must allow specifying pointers to instances (i.e., a pointer to a particular attribute in a row).

Fulfillment
The concept of a 'row' does not apply to SMI-DS, only to SMIv2, however OBJECT IDENTIFIER data objects can point to accessible SMIv2 tabular objects, and object names for SMIv2 tables do not change when translated to SMI-DS format.

4.1.20 Row Pointers [Yes]

Description
SMIng must allow specifying pointers to rows.

Fulfillment
The concept of a 'row' does not apply to SMI-DS, only to SMIv2, however OBJECT IDENTIFIER data objects can point to SMIv2 rows, and object names for SMIv2 tables do not change when translated to SMI-DS format.

4.1.21 Constraints on Pointers [Yes]

Description
SMIng must allow specifying the types of objects to which a pointer may point.

Fulfillment
A new variant of the SYNTAX clause is defined which restricts a particular data type that the OID pointer. E.g., "SYNTAX POINTER FooObject" or "SYNTAX POINTER InetAddress", would actually define an OBJECT IDENTIFIER.

4.1.22 Base Type Set [Yes]

Description
SMIng must support a fixed set of base types of fixed size and precision. The list of base types must not be extensible unless the SMI itself changes.

Fulfillment
SMI-DS uses a fixed set of base data types.

4.1.23 Extended Data Types [Yes]

Description
SMIng must support a mechanism to derive new types, which provide additional semantics (e.g., Counters, Gauges, Strings, etc.), from base types. It may be desirable to also allow the derivation of new types from derived types. New types must be as restrictive or more restrictive than the types that they are specializing.

Fulfillment
SMI-DS provides the TYPEDEF construct to specify complex or derived data types. LEAF definitions can derive attributes from a base type or another derived type.

4.1.24 Units, Formats, and Default Values of Defined Types and Attributes [Yes]

Description
In SMIv2 OBJECT-TYPE definitions may contain UNITS and DEFVAL clauses and TEXTUAL-CONVENTIONs may contain DISPLAY-HINTs. In a similar fashion units and default values must be applicable to defined types and format information must be applicable to attributes.

Fulfillment
SMI-DS retains the UNITS, DEFVAL, and DISPLAY-HINT clauses for all LEAF data type definitions and variable declarations.

4.1.25 Table Existence Relationships [Yes]

Description
SMIng must support INDEX, AUGMENTS, and EXTENDS in the SNMP/COPS-PR protocol mappings.

Fulfillment
These concepts have been included in SMI-DS, and AUGMENTS has been extended to any non-LEAF TYPEDEF. The EXTENDS construct is achieved by simply augmenting an existing ARRAY with a another (nested) ARRAY.

4.1.26 Table Existence Relationships [Yes]

Description
SMIng must support EXPANDS and REORDERS relationships in the SNMP/COPS-PR protocol mappings.

Fulfillment
SMI-DS is not a table-oriented data definition language like SMIv2 or SPPI. Aggregated data objects are defined in a nested manner to convey a hierarchical relationship. The EXPANDS and REORDERS clauses are only meaningful in this table-oriented framework. However, the DESCRIPTION clause is provided to express semantics such as EXPANDS and REORDERS.

4.1.27 Attribute Groups [Yes]

Description
An attribute group is a named, reusable set of attributes that are meaningful together. It can be reused as the type of attributes in other attribute groups (see also Section 4.1.28). This is similar to `structs' in C.

Fulfillment
SMI-DS provides the STRUCT macro for this purpose.

4.1.28 Containment [Yes]

Description
SMIng must provide support for the creation of new attribute groups from attributes of more basic types and potentially other attribute groups.

Fulfillment
SMI-DS allows arbitrary nesting of STRUCT, ARRAY, and UNION type definitions.

4.1.29 Single Inheritance [Yes]

Description
SMIng must provide support for mechanisms to extend attribute groups through single inheritance.

Fulfillment
SMI-DS allows new aggregate types to contain other aggregated types, by reference, i.e., the contained data object inherits all attributes from the type as defined in another TYPEDEF (and AUGMENTS, if any).

4.1.30 Reusable vs. Final Attribute Groups [Yes]

Description
SMIng must differentiate between "final" and reusable attribute groups, where the reuse of attribute groups covers inheritance and containment.

Fulfillment
SMI-DS provides the TYPEDEF macro to create reusable definitions, and variable declarations to identify 'final' attribute groups.

4.1.31 Events [Yes]

Description
SMIng must provide mechanisms to define events which identify significant state changes.

Fulfillment
The NOTIFICATION macro is used (slightly modified NOTIFICATION-TYPE macro.

4.1.32 Creation/Deletion [Maybe]

Description
SMIng must support a mechanism to define creation/deletion operations for instances. Specific creation/deletion errors, such as INSTALL-ERRORS, must be supported.

Fulfillment
A new data objected RowStatus could be defined, or the existing RowStatus simply used 'as-is' with data objects. This objective is very 'table-oriented' and protocol-specific. SMI-DS is intended to be protocol-independent.

4.1.33 Range and Size Constraints [Yes]

Description
SMIng must allow specifying range and size constraints where applicable.

Fulfillment
The SYNTAX clause is unchanged from SMIv2, which includes a range construct.

4.1.34 Uniqueness [Maybe]

Description
SMIng must allow the specification of uniqueness constraints on attributes. SMIng must allow the specification of multiple independent uniqueness constraints.

Fulfillment
Instance identifiers are of course unique. The DESCRIPTION clause is available to specify uniqueness characteristics for any LEAF data type or INDEX component.

4.1.35 Extension Rules [No]

Description
SMIng must provide clear rules how one can extend SMIng modules without causing interoperability problems "over the wire".

Fulfillment
The final version of SMI-DS will include a rigorous syntax, but there are no plans for an explicit EXTENSION construct, to allow SMI-DS to be extended in an distributed and uncontrolled manner. The SMI should only be changed in very careful and controlled manner, by an IETF WG (e.g., SMIng).

4.1.36 Deprecate Use of IMPLIED Keyword [Yes]

Description
The SMIng SNMP mapping must deprecate the use of the IMPLIED indexing schema.

Fulfillment
The IMPLIED keyword is deprecated in the SMI-DS INDEX construct.

4.1.37 No Redundancy [Yes]

Description
The SMIng language must avoid redundancy.

Fulfillment
SMI-DS remove any clause that is always the same value in all situations (e.g., MAX-ACCESS clause for the fooTable and fooEntry OBJECT-TYPE macros is always not-accessible, so only LEAF data objects have a MAX-ACCESS clause). The 'fooEntry' definition is removed entirely, and since SMI-DS is data object, not table oriented, there is no need for the ASN.1 'FooEntry SEQUENCE' construct. Basic containment relationships are implied by the aggregated data types themselves (nested ARRAY, UNION, STRUCT) rather than by using lots of verbose OBJECT-TYPE DESCRIPTION clauses to declare the containment relationships between various OBJECT-TYPE macros.

4.1.38 Compliance and Conformance [Yes]

Description
SMIng must provide a mechanism for compliance and conformance specifications for protocol-independent definitions as well as for protocol mappings.

Fulfillment
The SMI-DS module compliance section is unchanged from SMIv2. Just like SMIv2, only accessible (LEAF) objects are listed in this section.

4.1.39 Allow Refinement of All Definitions in Conformance Statements [Yes - TBD]

Description
SMIv2, RFC 2580, Section 3.1 says: <para removed> The last sentence forbids to put a not-accessible INDEX object into an OBJECT-GROUP. Hence, you can not refine its syntax in a compliance definition. For more details, see http://www.ibr.cs.tu-bs.de/ietf/smi-errata/.

Fulfillment
The arbitrary rules for SMIv2 can be changed, as they are adapted to SMI-DS. It is understood that every SMIv2 construct used in SMI-DS is subject to bugfixes.

4.1.40 Categories [No]

Description

SMIng must provide a mechanism to group definitions into subject categories. Concrete instances may only exist in the scope of a given subject category or context.

Fulfillment

SMI-DS currently has no such construct. This would require management and coordination of the set of categories, and therefore further thought. Such a construct could be added if required.

4.1.41 Core Language Keywords vs. Defined Identifiers [No - TBD]

Description

In SMI and SPPI modules some language keywords (macros and a number of basetypes) have to be imported from different SMI language defining modules, e.g., OBJECT-TYPE, MODULE-IDENTITY, Integer32 must to be imported from SNMPv2-SMI and TEXTUAL- CONVENTION must be imported from SNMPv2-TC, if used. MIB authors are continuously confused about these import rules. In SMIng only defined identifiers must be imported. All SMIng language keywords must be implicitly known and there must not be a need to import them from any module.

Fulfillment

Currently, the SMI-DS IMPORTS clause is unchanged from SMIv2. It would be a mistake to forbid IMPORTS of base data types, since this is just one more thing for authors to get wrong. The burden of listing all external definitions, including base types, in the IMPORTS clause is not a problem worth solving. The SMI-DS rules could be changed to make IMPORTS of base types forbidden, optional, or mandatory, whatever is required.

4.1.42 Instance Naming [Maybe - TBD]

Description

Instance naming in SMIv2 and SPPI is different. SMIng must align the instance naming (either in the protocol neutral model or the protocol mappings).

Fulfillment

SMI-DS instance naming is compatible with SMIv2. It is not clear what additions are needed to support SPPI naming as well.

4.1.43 Length of Identifiers [Yes - TBD]

Description
The allowed length of the various kinds of identifiers must be extended from the current `should not exceed 32' (maybe even from the `must not exceed 64') rule.

Fulfillment
All the arbitrary SMIv2 rules are subject to removal or repair as they are transferred to SMI-DS. The maximum descriptor length an agent must accept will be extended to 64.

4.1.44 Assign OIDs in the Protocol Mappings [No]

Description
SMIng must not assign OIDs to reusable definition of attributes, attribute groups, events, etc. Instead, SNMP and COPS-PR mappings must assign OIDs to the mapped items.

Fulfillment
Although TYPEDEF definitions actually meet this requirement because only variable declarations can have complete OID assignments, it would be a critical mistake to separate data object naming from the data definition itself. There is no justification whatsoever for the management transport protocol to dictate the naming characteristics of the data definition language.

4.2.1 Methods [No]

Description
SMIng should support a mechanism to define method signatures (parameters, return values, exception) that are implemented on agents.

Fulfillment
SMI-DS defines a data definition language with sufficient power to be used as a platform for object-oriented network management definitions in the future (ala C --> C++ transition).

4.2.2 Unions [Yes]

Description
Allows an attribute to contain one of many types of values. The lack of unions has also lead to relatively complex sparse table work-around in some DISMAN mid-level managers. Despite from

discriminated unions (see Section 4.1.18), this kind of union has no accompanied explicit discriminator attribute that selects the union's type of value.

Fulfillment
SMI-DS provides the UNION macro for this purpose.

4.2.3 Float Data Types [Yes]

Description
SMIng should support the base data types Float32, Float64, Float128.

Fulfillment
SMI-DS will support a Float data type. Is is not clear that 3 variants are needed though.

4.2.4 Comments [Yes]

Description
The syntax of comments should be well defined, unambiguous and intuitive to most people, e.g., the C++/Java `//' syntax.

Fulfillment
The ASN.1 comment meets these requirements and is used unchanged from SMIv2. There is no community requirement to use Java style comments. The use of 2 dashes for a 'start of comment' token is not any better or worse than 2 slashes. Not a change worth making.

4.2.5 Referencing Tagged Rows [No]

Description
PIB and MIB row attributes reference a group of entries in another table. SPPI formalizes this by introducing PIB-TAG and PIB-REFERENCES clauses. This functionality should be retained in SMIng.

Fulfillment
SMI-DS does not use a table-oriented data model, so these constructs do not apply.

4.2.6 Arrays [Yes]

Description
SMIng should allow the definition of a SEQUENCE OF attributes or

attribute groups (Section 4.1.27).

Fulfillment
SMI-DS provides the ARRAY macro for this purpose.

4.2.7 Internationalization [No - TBD]

Description
Informational text (DESCRIPTION, REFERENCE, ...) should allow i18nized encoding, probably UTF-8.

Fulfillment
SMI-DS used the DESCRIPTION and REFERENCE clauses unchanged from SMIv2. Changes to these clauses could be made if required, but unless standard (IETF) information modules are written in a language other than English, this only applies to vendor MIBs.

4.2.8 Separate Data Modelling from Management Protocol Mapping [Yes]

Description
It should be possible to separate the domain specific data modelling work from the network management protocol specific work.

Fulfillment
The SMI-DS data definitions are protocol independent. Mappings (where applicable) will be defined for SNMP, because SMIv2 is intended to function with SNMP, and SMI-DS is intended to replace SMIv2. Mapping rules for other protocols are certainly possible, but are not included in this document.

12. Security Considerations

This document defines a structure for management data and therefore does not expose any management information from a particular device. However, accessible data objects defined with the mechanisms defined in this document should be given the same security consideration as objects specified with SMIv2, when being transferred with SNMP.

SNMPv1 by itself is not a secure environment. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model RFC 2574 [RFC2574] and the View-based Access Control Model RFC 2575 [RFC2575] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to an instance of this MIB, is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

13. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property 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; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication 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 implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

14. Acknowledgements

Portions of the existing SMI RFCs, SMIng drafts, and the ANSI C Programming Language inspired many of the concepts discussed in this memo.

15. Normative References

[RFC1905]
SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, SNMP Research, Inc., Cisco Systems, Inc., Dover Beach Consulting, Inc., International Network Services, January 1996.

[RFC1906]
SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, SNMP Research, Inc., Cisco Systems, Inc., Dover Beach Consulting, Inc., International Network Services, January 1996.

[RFC2026]
Bradner, S., "The Internet Standards Process -- Revision 3", RFC 2026, Harvard University, October, 1996.

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

[RFC2571]
Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, Cabletron Systems, Inc., BMC Software, Inc., IBM T. J. Watson Research, April 1999.

[RFC2572]
Case, J., Harrington D., Presuhn R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, SNMP Research, Inc., Cabletron Systems, Inc., BMC Software, Inc., IBM T. J. Watson Research, April 1999.

[RFC2573]
Levi, D., Meyer, P., and B. Stewart, "SNMPv3 Applications", RFC 2573, SNMP Research, Inc., Secure Computing Corporation, Cisco Systems, April 1999.

[RFC2574]
Blumenthal, U., and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, IBM T. J. Watson Research, April 1999.

[RFC2575]
Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2575, IBM T. J. Watson Research, BMC Software, Inc., Cisco Systems, Inc., April 1999.

[RFC2578]
McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", RFC 2578, STD 58, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, April 1999.

[RFC2579]
McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, "Textual Conventions for SMIv2", RFC 2579, STD 58, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, April 1999.

[RFC2580]
McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, "Conformance Statements for SMIv2", RFC 2580, STD 58, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, April 1999.

16. Informative References

[DSMON-MIB]
Bierman, A., "Remote Monitoring MIB Extensions for Differentiated Services", Work in progress (draft-ietf-rmonmib-dsmon-mib-09.txt), Cisco Systems, Inc., November 2001.

[RFC1155]
Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[RFC1157]
Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[RFC1212]
Rose, M., and K. McCloghrie, "Concise MIB Definitions", RFC 1212,

Performance Systems International, Hughes LAN Systems, March 1991.

[RFC1215]
M. Rose, "A Convention for Defining Traps for use with the SNMP", RFC 1215, Performance Systems International, March 1991.

[RFC1901]
SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, SNMP Research, Inc., Cisco Systems, Inc., Dover Beach Consulting, Inc., International Network Services, January 1996.

[RFC2021]
S. Waldbusser, "Remote Network Monitoring Management Information Base Version 2 using SMIv2", RFC 2021, INS, January 1997.

[RFC2570]
Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, SNMP Research, Inc., TIS Labs at Network Associates, Inc., Ericsson, Cisco Systems, April 1999.

[RFC2737]
McCloghrie, K., and A. Bierman, "Entity MIB (Version 2)", RFC 2737, Cisco Systems, Inc., December 1999.

[RFC2851]
Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 2851, Compaq Computer Corporation, Nortel Networks, Wind River Systems, Inc., TU Braunschweig, June 2000.

[RFC2863]
McCloghrie, K., and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, Cisco Systems, Argon Networks, June, 2000.

[RFC3216]
Elliot, C., Harrington, D., Jason, J., Schoenwaelder, J., Strauss, F., and W. Weiss, "SMIng Objectives", RFC 3216, Cisco Systems, Enterasys Networks, Intel Corporation, TU Braunschweig, Ellacoya Networks, December 2001.

17. Author's Address

Andy Bierman
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA USA 95134
Phone: +1 408-527-3711
Email: abierman@cisco.com

18. Full Copyright Statement