Translation of SMIv2 MIB Modules to YANG Modules
draft-schoenw-netmod-smi-yang-01

Abstract

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol, NETCONF remote procedure calls, and NETCONF notifications. The Structure of Management Information (SMIv2) defines fundamental data types, an object model, and the rules for writing and revising MIB modules for use with the SNMP protocol. This document defines a translation of SMIv2 MIB modules into YANG modules.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 12, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.iert.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction .................................................. 3
2. Translation of SMIv2 to YANG ................................. 4
   2.1. Mapping of Special Types .................................. 4
   2.2. Module Prefix Generation ................................. 5
   2.3. Translation of SMIv2 Modules ............................. 5
   2.4. Translation of SMIv2 Imports ............................. 6
   2.5. Translation of the MODULE-IDENTITY Macro ................ 7
   2.6. Translation of the TEXTUAL-CONVENTION Macro ............ 7
   2.7. Translation of OBJECT IDENTIFIER Assignments ........... 8
   2.8. Translation of the OBJECT-TYPE Macro .................... 8
   2.8.1. Translation scalars and columnar objects .............. 8
   2.8.2. Translation tree nodes and non-augmenting conceptual tables .................................. 9
   2.8.3. Translation augmenting conceptual tables ............. 10
   2.9. Translation of the OBJECT-IDENTITY Macro ............... 10
   2.10. Translation of the NOTIFICATION-TYPE Macro ............ 10
3. YANG Extension Definition ..................................... 12
4. IANA Considerations ........................................... 14
5. Security Considerations ....................................... 15
6. References ..................................................... 16
   6.1. Normative References ..................................... 16
   6.2. Informative References ................................... 16
Appendix A. Example: Translation of IF-MIB ....................... 17
Appendix B. Changes from 00 to 01 ............................... 24
Author’s Address ............................................... 25
1. Introduction

This document describes an translation of SMIv2 [RFC2578], [RFC2579], [RFC2580] MIB modules into YANG [RFC6020] modules. The mapping is illustrated in Appendix A by outlining the translation of the IF-MIB [RFC2863] SMIv2 module.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].
2. Translation of SMIV2 to YANG

This section defines the mapping of SMIV2 constructs to YANG constructs.

2.1. Mapping of Special Types

The SMIV2 base types and some well known derived textual-conventions are mapped to YANG types according to Table 1. The mapping of the OCTET STRING depends on the context. If an OCTET STRING type has an associated DISPLAY-HINT, then the corresponding YANG base type is the string type. Otherwise, the binary type is used. Similarly, the mapping of the INTEGER type depends on its usage as an enumeration or a 32-bit integral type.

<table>
<thead>
<tr>
<th>SMIV2 Module</th>
<th>SMIV2 Type</th>
<th>YANG Module</th>
<th>YANG Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPv2-SMI</td>
<td>INTEGER</td>
<td></td>
<td>enumeration</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>INTEGER</td>
<td></td>
<td>int32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Integer32</td>
<td></td>
<td>int32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>OCTET STRING</td>
<td></td>
<td>binary</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>OCTET STRING</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>OBJECT</td>
<td>ietf-yang-types</td>
<td>object-identifier</td>
</tr>
<tr>
<td></td>
<td>IDENTIFIER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>BITS</td>
<td></td>
<td>bits</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>IpAddress</td>
<td>ietf-inet-types</td>
<td>ipv4-address</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Counter32</td>
<td>ietf-yang-types</td>
<td>counter32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Gauge32</td>
<td>ietf-yang-types</td>
<td>gauge32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>TimeTicks</td>
<td>ietf-yang-types</td>
<td>timeticks</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Opaque</td>
<td></td>
<td>binary</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Counter64</td>
<td>ietf-yang-types</td>
<td>counter64</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Unsigned32</td>
<td></td>
<td>uint32</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>PhysAddress</td>
<td>ietf-yang-types</td>
<td>phys-address</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>MacAddress</td>
<td>ietf-yang-types</td>
<td>mac-address</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>TimeStamp</td>
<td>ietf-yang-types</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

Table 1

Note that the mappings shown above may impact the imports of a module. Implementations must add any additional imports required by the mapping.
2.2. Module Prefix Generation

The input to the prefix generation algorithm is a set of prefixes (usually derived from imported module names) and a specific module name to convert into a prefix. The algorithm described below produces a prefix for the given module name that is unique within the set of prefixes.

**Special prefixes for well known SMIv2 and YANG modules**

<table>
<thead>
<tr>
<th>YANG / SMIv2 Module</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>ietf-yang-types</td>
<td>yang</td>
</tr>
<tr>
<td>ietf-inet-types</td>
<td>inet</td>
</tr>
</tbody>
</table>

Table 2

- First, some fixed translations (see Table 2) mapping well known SMIv2 and YANG modules to short prefixes are tried. If a fixed translation rule exists and leads to a conflict free prefix, then the result of the fixed translation is used.

- Otherwise, prefixes are generated by tokenizing an SMIv2 module name where hyphens are treated as token separators. The tokens associated with a module name are converted to lowercase characters. The shortest sequence of token concatenated using hyphens as separators which includes at least two tokens and is unique among all prefixes used in the set of prefixes associated with module names.

In the worst case, the prefix derived from an SMIv2 module name becomes the SMIv2 module name translated to lower-case. But on average, much shorter prefixes are generated.

2.3. Translation of SMIv2 Modules

SMIv2 modules are mapped to corresponding YANG modules. The YANG module name is the same as the SMIv2 module name.

The YANG namespace is constructed out of a constant prefix followed by the SMIv2 module name. Since SMIv2 module names are unique, the resulting YANG namespace is unique. The registered prefix is urn:ietf:params:xml:ns:yang:smi:v2:, see the IANA considerations section.

The YANG prefix is derived from the SMIv2 module name. Since the
YANG prefix is supposed to be short and must be unique within the set of all prefixes used by a YANG module, the module prefix generation algorithm described in Section 2.2 is used.

2.4. Translation of SMIv2 Imports

SMIv2 IMPORT clauses are translated to YANG import statements. One major difference between the SMIv2 import mechanism and the YANG import mechanism is that SMIv2 imports specific symbols from a module while the YANG import statement imports all symbols of the referenced YANG module.

SMIv2 imports that are ignored in YANG

<table>
<thead>
<tr>
<th>SMIV2 Module</th>
<th>SMIV2 Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMPv2-SMI</td>
<td>MODULE-IDENTITY</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>OBJECT-IDENTITY</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>OBJECT-TYPE</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>NOTIFICATION-TYPE</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>mib-2</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>TEXTUAL-CONVENTION</td>
</tr>
<tr>
<td>SNMPv2-CONF</td>
<td>OBJECT-GROUP</td>
</tr>
<tr>
<td>SNMPv2-CONF</td>
<td>NOTIFICATION-GROUP</td>
</tr>
<tr>
<td>SNMPv2-CONF</td>
<td>MODULE-COMPLIANCE</td>
</tr>
<tr>
<td>SNMPv2-CONF</td>
<td>AGENT-CAPABILITIES</td>
</tr>
<tr>
<td>SNMPv2-MIB</td>
<td>snmpTraps</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>all symbols</td>
</tr>
<tr>
<td>SNMPv2-CONF</td>
<td>all symbols</td>
</tr>
</tbody>
</table>

Table 3

In order to produce correct and complete YANG import statements, it is necessary to apply the following rules:

- Ignore all imports listed in Table 3. Note that the modules SNMPv2-SMI and SNMPv2-CONF are completely ignored since all definitions in these modules are translated by translation rules.

- Add any imports required by the type translations according to the type mapping table. This requires to consider all the types used in the translation unit.

The argument of the generated import statements are the untranslated SMIv2 module name. The import statement must contain a prefix statement. The prefixes are generated by applying the module prefix
2.5. Translation of the MODULE-IDENTITY Macro

The clauses of the SMIv2 MODULE-IDENTITY macro are mapped to equivalent YANG statements.

- The SMIv2 ORGANIZATION clause is mapped to the YANG organization statement.
- The SMIv2 CONTACT-INFO clause is mapped to the YANG contact statement.
- The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.
- Each SMIv2 REVISION clause is mapped to a YANG revision statement. The revision is identified by the date contained in the SMIv2 REVISION. DESCRIPTION sub-clauses of REVISION clauses are mapped to corresponding description statement nested in revision clauses.
- The SMIv2 LAST-UPDATED is ignored if the associated date matches a REVISION clause. Otherwise, an additional revision statement is generated.
- The value of the invocation of an SMIv2 MODULE-IDENTITY macro is ignored.

2.6. Translation of the TEXTUAL-CONVENTION Macro

The SMIv2 uses invocations of the TEXTUAL-CONVENTION macro to define new types derived from the SMIv2 base types. Invocations of the TEXTUAL-CONVENTION macro are translated into YANG typedef statements.

The name of the TEXTUAL-CONVENTION macro invocation is used as the name of the generated typedef statement. The clauses of the SMIv2 TEXTUAL-CONVENTION macro are mapped to YANG statements embedded in the typedef statement as follows:

- The SMIv2 DISPLAY-HINT clause is used to determine the type mapping of types derived form the OCTET STRING type as explained in Section 2.1. Furthermore, the DISPLAY-HINT value is used to generate a regular expression for the YANG pattern statement within the type statement. [[TODO: Define a translation algorithm that is simple and produces correct and usable results for the majority of simple DISPLAY-HINTS.]]
o The SMIv2 STATUS clause is mapped to the YANG status statement. The generation of the YANG status statement is skipped if the value of the STATUS clause is current.

o The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.

o The SMIv2 REFERENCE clause is mapped to the YANG reference statement.

o The SMIv2 SYNTAX clause is mapped to the YANG type statement.

SMIv2 range restrictions are mapped to YANG range statements while SMIv2 length restrictions are mapped to YANG length statements.

SMIv2 INTEGER enumerations and SMIv2 BITS are mapped to YANG enum / value and bit / position statements.

2.7. Translation of OBJECT IDENTIFIER Assignments

The mapping suppresses many structural OBJECT IDENTIFIER assignments that are typically used to organize the OBJECT IDENTIFIER tree. Only tree nodes that contain scalars or tables are translated to YANG containers. When generating YANG containers, the container is marked as config false. [[DISCUSS: Do we keep this flat translation? Are there corner cases where structural OIDs are used as values?]]

2.8. Translation of the OBJECT-TYPE Macro

The SMIv2 uses the OBJECT-TYPE macro to define objects and the structure of conceptual tables. Objects exist either as scalars (exactly one instance within an SNMP context) or columnar objects (zero or multiple instances within an SNMP context) within conceptual tables. A subset of columnar objects of a table define the index (key) of the table. Furthermore, conceptual tables can augment other conceptual tables. All these differences must be taken into account when mapping SMIv2 OBJECT-TYPE macro invocations to YANG.

2.8.1. Translation scalars and columnar objects

The SMIv2 OBJECT-TYPE macro invocations defining scalars or columnar objects are translated to YANG leaf statements. The name of the leaf is the name associated with the SMIv2 OBJECT-TYPE macro invocation.

o The SMIv2 SYNTAX clause is mapped to the YANG type clause. Embedded clauses are generates as described in Section 2.1.

o The SMIv2 UNITS clause is mapped to the YANG units statement.
2.8.2. Translation tree nodes and non-augmenting conceptual tables

In a first pass, iterate over the object identifier tree identifying all nodes that contain scalars and all non-augmenting conceptual tables. For each node, create a YANG container statement. For nodes representing groups of scalar objects, generate the necessary YANG leaf statements as described above. For nodes representing non-augmenting conceptual tables, identify the table entry OBJECT-TYPE, create a YANG list statement named after the SMIv2 entry OBJECT-TYPE. The rest of the clauses are translated as follows:

- The SMIv2 SYNTAX clause is ignored.
- The SMIv2 UNITS clause is ignored.
- The SMIv2 MAX-ACCESS clause is ignored.
- The SMIv2 STATUS clause is mapped to the YANG status statement. The generation of the YANG status statement is skipped if the value of the STATUS clause is current.
- The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.
- The SMIv2 REFERENCE clause is mapped to the YANG reference statement.
- The value of the SMIv2 OBJECT-TYPE macro invocation is ignored.

Note that the SMIv2 non-augmenting conceptual table node is not translated to YANG. Within the list statement, create YANG leaf
nodes as described above. For objects listed in the SMIv2 INDEX clause that are not part of the conceptual table itself, create YANG leaf statements of type leafref pointing to the referenced definition.

2.8.3. Translation augmenting conceptual tables

In a second pass, iterate over all augmenting conceptual tables. For each augmenting conceptual table, identify the table entry OBJECT-TYPE, create a YANG augment statement with the first argument containing the path of the augmented table. The rest of the clauses are translated as follows:

- The SMIv2 SYNTAX clause is ignored.
- The SMIv2 UNITS clause is ignored.
- The SMIv2 MAX-ACCESS clause is ignored.
- The SMIv2 STATUS clause is mapped to the YANG status statement. The generation of the YANG status statement is skipped if the value of the STATUS clause is current.
- The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.
- The SMIv2 REFERENCE clause is mapped to the YANG reference statement.
- The value of the SMIv2 OBJECT-TYPE macro invocation is ignored.

Note that the SMIv2 augmenting conceptual table node is not translated to YANG. Within the augment statement, create YANG leaf nodes as described above.

2.9. Translation of the OBJECT-IDENTITY Macro

[[TODO]]

2.10. Translation of the NOTIFICATION-TYPE Macro

The SMIv2 provides the NOTIFICATION-TYPE macro to define notifications. YANG provides the notification statement for the same purpose. The name of the NOTIFICATION-TYPE macro invocation is used as the name of the generated notification statement. The clauses of the NOTIFICATION-TYPE macro are mapped to YANG statements embedded in the notification statement as follows.
o The SMIv2 OBJECTS clause is mapped to a sequence of YANG containers. For each object listed in the OBJECTS clause value, a YANG container statement is generated. The name of this container is the name of the notification and the name of the current concept concatenated by a hyphen. If the current object belongs a conceptual table, then a sequence of leaf statements is generated for each INDEX of the SMIv2 conceptual table. Next, a leaf statement is generated for the current object. All container leafs are marked as config false.

o The SMIv2 STATUS clause is mapped to the YANG status statement. The generation of the YANG status statement is skipped if the value of the STATUS clause is current.

o The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.

o The SMIv2 REFERENCE clause is mapped to the YANG reference statement.

o The value of the SMIv2 NOTIFICATION-TYPE macro invocation is ignored.
3. YANG Extension Definition

This section defines some YANG extension statements that can be used to carry additional information from the original SMIv2 module into the YANG module. The YANG module references [RFC2578] and [RFC2579].

```yaml
<CODE BEGINS> file "ietf-yang-smiv2@2010-11-08.yang"

module ietf-yang-smiv2 {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-smiv2";
  prefix "smiv2";

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/netmod/>
              WG List:  <mailto:netmod@ietf.org>
              WG Chair:  David Kessens
                          <mailto:david.kessens@nsn.com>
              WG Chair:  Juergen Schoenwaelder
                          <mailto:j.schoenwaelder@jacobs-university.de>
              Editor:    Juergen Schoenwaelder
                          <mailto:j.schoenwaelder@jacobs-university.de>";

  description
    "This module defines YANG extensions that are used to translate
     SMIv2 concepts into YANG."

  Copyright (c) 2010 IETF Trust and the persons identified as
  authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, is permitted pursuant to, and subject to the license
  terms contained in, the Simplified BSD License set forth in Section 4.c
  of the IETF Trust’s Legal Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see the
  RFC itself for full legal notices.";

<CODE ENDS>
```
revision 2010-09-24 {
  description
  "Initial revision.";
  reference
  "RFC XXXX: Translation of SMIv2 MIB Modules to YANG Modules";
  // RFC Ed.: replace XXXX with actual RFC number and remove this note
}

extension oid {
  argument "value";
  description
  "The oid statement takes as an argument the object identifier
   assigned to an SMIv2 definition. The object identifier value
   is written in decimal dotted notation.";
  reference
  "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}

extension display-hint {
  argument "format";
  description
  "The display-hint statement takes as an argument the DISPLAY-HINT
   assigned to an SMIv2 textual convention.";
  reference
  "RFC2579: Textual Conventions for SMIv2";
}

extension max-access {
  argument "access";
  description
  "The max-access statement takes as an argument the MAX-ACCESS
   assigned to an SMIv2 object definition";
  reference
  "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}

extension defval {
  argument "value";
  description
  "The defval statement takes as an argument a default value defined
   by an SMIv2 DEFVAL clause.";
  "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}

}<CODE ENDS>
4. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registrations have been made.

Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name:         ietf-yang-smiv2
prefix:       smiv2
reference:    RFC XXXX
5. Security Considerations

TBD
6. References

6.1. Normative References


6.2. Informative References


Appendix A. Example: Translation of IF-MIB

The translation of the IF-MIB [RFC2863] leads to the following YANG module frame. The prefix is the translation of the SMIv2 module name IF-MIB to lowercase (consisting of two token and thus no abbreviation).

```yaml
module IF-MIB {
    prefix "if-mib";
}
```

The translation of the IMPORTs of the IF-MIB leads to the following YANG import statements:

```yaml
import IANAifType-MIB      { prefix "ianaiftype-mib"; }
import SNMPv2-TC           { prefix "smiv2-tc"; }
import ietf-yang-types     { prefix "yang"; }
import ietf-yang-smiv2     { prefix "smiv2"; }
```

The translation of the IF-MIB MODULE-IDENTITY macro invocation leads to the following YANG statements:
The translation of the OwnerString and InterfaceIndex textual-conventions of the IF-MIB are shown below.
typedef OwnerString {
  type string {
    length "0..255";
    pattern "\p{IsBasicLatin}{0,255}";
  }
  status deprecated;
  description
  "This data type is used to model an administratively assigned name of the owner of a resource. This information is taken from the NVT ASCII character set. It is suggested that this name contain one or more of the following: ASCII form of the manager station’s transport address, management station name (e.g., domain name), network management personnel’s name, location, or phone number. In some cases the agent itself will be the owner of an entry. In these cases, this string shall be set to a string starting with ‘agent’.";
}

typedef InterfaceIndex {
  type int32 {
    range "1..2147483647";
  }
  description
  "A unique value, greater than zero, for each interface or interface sub-layer in the managed system. 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.";
}

The translation of the some key parts of the IF-MIB are shown below.

container interfaces {
  config false;

  leaf ifNumber {
    type int32;
    description
    "The number of network interfaces (regardless of their current state) present on this system.";
  }

  list ifEntry {
    key "ifIndex";
    description
    "An entry containing management information applicable to a
leaf ifIndex {
  type if-mib:InterfaceIndex;
  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.";
}

leaf ifDescr {
  type smiv2-tc:DisplayString {
    length "0..255";
  }
  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.";
}

/* skipping several definitions */

leaf ifAdminStatus {
  type enumeration {
    enum up      { value 1; }
    enum down    { value 2; }
    enum testing { value 3; }
  }
  description
  "The desired state of the interface. The testing(3) state
  indicates that no operational packets can be passed. When a
  managed system initializes, all interfaces start with
  ifAdminStatus in the down(2) state. As a result of either
  explicit management action or per configuration information
  retained by the managed system, ifAdminStatus is then
  changed to either the up(1) or testing(3) states (or remains
  in the down(2) state).";
}

/* skipping more definitions */

}
augment "/if-mib:interfaces/if-mib:ifEntry" {
    description
    "An entry containing additional management information applicable to a particular interface.";

    leaf ifName {
        type smiv2-tc:DisplayString;
        description
        "The textual name of the interface. The value of this object should be the name of the interface as assigned by the local device and should be suitable for use in commands entered at the device’s ‘console’. This might be a text name, such as ‘le0’ or a simple port number, such as ‘1’, depending on the interface naming syntax of the device. If several entries in the ifTable together represent a single interface as named by the device, then each will have the same value of ifName. Note that for an agent which responds to SNMP queries concerning an interface on some other (proxied) device, then the value of ifName for such an interface is the proxied device’s local name for it.

        If there is no local name, or this object is otherwise not applicable, then this object contains a zero-length string.";
    }

    /* skipping more definitions */
}

The translation of the linkDown notification of the IF-MIB is shown below.

notification linkDown {
    description
    "A linkDown trap signifies that the SNMP entity, acting in an agent role, has detected that the ifOperStatus object for one of its communication links is about to enter the down state from some other state (but not from the notPresent state). This other state is indicated by the included value of ifOperStatus.";

    container linkDown-ifIndex {
        config false;
        leaf ifIndex {
            type leafref {
                path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
            }
            description
            "[Automagically generated leafref leaf.]";
        }
    }
}
```

container linkDown-ifAdminStatus {
  config false;
  leaf ifIndex {
    type leafref {
      path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
    }
    description
      "[Automagically generated leafref leaf.];"
  }
  leaf ifAdminStatus {
    type enumeration {
      enum up { value 1; }
      enum down { value 2; }
      enum testing { value 3; }
    }
    description
      "The desired state of the interface. The testing(3) state
indicates that no operational packets can be passed. When a
managed system initializes, all interfaces start with
ifAdminStatus in the down(2) state. As a result of either
explicit management action or per configuration information
retained by the managed system, ifAdminStatus is then
changed to either the up(1) or testing(3) states (or remains
in the down(2) state).";
  }
}

container linkDown-ifOperStatus {
  config false;
  leaf ifIndex {
    type leafref {
      path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
    }
    description
      "[Automagically generated leafref leaf.];"
  }
  leaf ifOperStatus {
    type enumeration {
      enum up { value 1; }
      enum down { value 2; }
      enum testing { value 3; }
      enum unknown { value 4; }
      enum dormant { value 5; }
      enum notPresent { value 6; }
      enum lowerLayerDown { value 7; }
    }
```
description
"The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed. If ifAdminStatus is down(2) then ifOperStatus should be down(2). If ifAdminStatus is changed to up(1) then ifOperStatus should change to up(1) if the interface is ready to transmit and receive network traffic; it should change to dormant(5) if the interface is waiting for external actions (such as a serial line waiting for an incoming connection); it should remain in the down(2) state if and only if there is a fault that prevents it from going to the up(1) state; it should remain in the notPresent(6) state if the interface has missing (typically, hardware) components.";
Appendix B. Changes from 00 to 01

- Translation is config false; top-level container are marked as config false.
- Revised the overall document structure, added a YANG module for the definition of YANG extensions (smiv2:oid, smiv2:display-hint, smiv2:max-access, smiv2:defval), moved the IF-MIB example into an appendix.
- Alignment with RFC 6020 and RFC 6021.
- Started to use [[DISCUSS]] and [[TODO]] markers inside the text instead of maintaining a TODO list as an appendix.
Author’s Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de