Translation of SMIv2 MIB Modules to YANG Modules

draft-schoenw-netmod-smi-yang-00

Status of this Memo

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

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.

This Internet-Draft will expire on August 3, 2009.

Copyright Notice

Copyright (c) 2009 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.ietf.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.
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. This document describes the translation of SMIv2 MIB modules into YANG modules.

Table of Contents

1. Introduction ...................................................... 3
2. Translation of Special Types ................................. 4
3. Module Prefix Generation .................................... 5
4. Mapping of SMIv2 Modules ................................... 6
5. Mapping of SMIv2 Imports .................................... 7
6. Mapping of the MODULE-IDENTITY Macro ..................... 9
7. Mapping of the TEXTUAL-CONVENTION Macro ................. 11
8. Mapping of the OBJECT-TYPE Macro .......................... 13
   8.1. Translating scalars and columnar objects ............... 13
   8.2. Translating tree nodes and non-augmenting conceptual tables .......................... 13
   8.3. Translating augmenting conceptual tables .............. 14
   8.4. Translation of IF-MIB objects ............................ 15
9. Mapping of the NOTIFICATION-TYPE Macro ................... 18
10. IANA Considerations ........................................ 21
11. Security Considerations .................................... 22
12. References .................................................... 23
   12.1. Normative References ................................ 23
   12.2. Non-Normative References ................................ 23
Appendix A. Open Issues ........................................ 24
Author's Address ................................................. 25
1. Introduction

This document describes a translation of SMIv2 [RFC2578], [RFC2579], [RFC2580] MIB modules into YANG modules. The mapping is illustrated by considering the IF-MIB [RFC2863] as an example SMIv2 module and its translation.

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 Special Types

The SMIv2 base types and some well known derived textual-conventions are mapped to YANG types according to the table shown below. 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.

Mapping of SMIv2 types to YANG types

<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>yang-types</td>
<td>object-identifier</td>
</tr>
<tr>
<td></td>
<td>IDENTIFIER</td>
<td></td>
<td>string</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>inet-types</td>
<td>ipv4-address</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Counter32</td>
<td>yang-types</td>
<td>counter32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>Gauge32</td>
<td>yang-types</td>
<td>gauge32</td>
</tr>
<tr>
<td>SNMPv2-SMI</td>
<td>TimeTicks</td>
<td>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>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>yang-types</td>
<td>phys-address</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>MacAddress</td>
<td>ieee-types</td>
<td>mac-address</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>TimeStamp</td>
<td>yang-types</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

Note that the mappings shown above may impact the imports of a module. Implementations must add any additional imports required by the mapping.
3. 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>yang-types</td>
<td>yang</td>
</tr>
<tr>
<td>inet-types</td>
<td>inet</td>
</tr>
<tr>
<td>ieee-types</td>
<td>ieee</td>
</tr>
<tr>
<td>SNMPv2-TC</td>
<td>smiv2</td>
</tr>
</tbody>
</table>

- First, some fixed translations (see Table above) 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.
4. Mapping 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 suggested prefix is urn:ietf:params:xml:ns:yang:smiv2: but usage of this prefix requires a specification and an associated IETF/IANA allocation action.

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 3 is used.

The translation of the IF-MIB 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";

}
```
5. Mapping 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>

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

- Ignore all imports listed in the table above. 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 generation algorithm described in Section 3.

The translation of the IMPORTs of the IF-MIB leads to the following YANG import statements:
import IANAifType-MIB { prefix "ianaiftype-mib"; }
import SNMPv2-TC { prefix "smiv2"; }
import yang-types { prefix "yang"; }
6. Mapping of the MODULE-IDENTITY Macro

The clauses of the SMIPv2 MODULE-IDENTITY macro are mapped to equivalent YANG statements.

- The SMIPv2 ORGANIZATION clause is mapped to the YANG organization statement.
- The SMIPv2 CONTACT-INFO clause is mapped to the YANG contact statement.
- The SMIPv2 DESCRIPTION clause is mapped to the YANG description statement.
- Each SMIPv2 REVISION clause is mapped to a YANG revision statement. The revision is identified by the date contained in the SMIPv2 REVISION. DESCRIPTION sub-clauses of REVISION clauses are mapped to corresponding description statement nested in revision clauses.
- The SMIPv2 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 SMIPv2 MODULE-IDENTITY macro is ignored.

The translation of the IF-MIB MODULE-IDENTITY macro invocation leads to the following YANG statements:
organization
"IETF Interfaces MIB Working Group";

contact
"Keith McCloghrie
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA  95134-1706
US
408-526-5260
kzm@cisco.com";

description
"The MIB module to describe generic objects for network interface sub-layers. This MIB is an updated version of MIB-II's ifTable, and incorporates the extensions defined in RFC 1229.";

revision "2000-06-14" {
  description
  "Clarifications agreed upon by the Interfaces MIB WG, and published as RFC 2863.";
}

revision "1996-02-28" {
  description
  "Revisions made by the Interfaces MIB WG, and published in RFC 2233.";
}

revision "1993-11-08" {
  description
  "Initial revision, published as part of RFC 1573.";
}
7. Mapping 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 from the OCTET STRING type as explained in Section 2. Furthermore, the DISPLAY-HINT value is used to generate a regular expression for the YANG pattern statement within the type statement. The current implementation uses libsmi’s DISPLAY-HINT to regular expression translation algorithm.

- 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 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.

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.";
}
8. Mapping 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.

8.1. Translating 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.

- The SMIv2 SYNTAX clause is mapped to the YANG type clause. Embedded clauses are generates as described in Section 2.
- The SMIv2 UNITS clause is mapped to the YANG units statement.
- The SMIv2 MAX-ACCESS clause is mapped to the YANG config statement with the value false if the value of the MAX-ACCESS clause is not read-write or read-create, or the context of the leaf requires it to be false.
- 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.

8.2. Translating 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 SMIv2 INDEX clause is mapped to the YANG key clause listing the columnar objects forming the key of the YANG list.
- 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 keyref pointing to the referenced definition.

### 8.3. Translating 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.

8.4. Translation of IF-MIB objects

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

```yml
container interfaces {
  leaf ifNumber {
    type int32;
    config false;
    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 particular interface.";

    leaf ifIndex {
      type if-mib:InterfaceIndex;
      config false;
      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:DisplayString {
        length "0..255";
      }
    }
  }
}
```
config false;

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."
;

leaf ifAdminStatus {
    type enumeration {
        enum up { value 1; }
        enum down { value 2; }
        enum testing { value 3; }
    }
    config true;

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:DisplayString;
    config false;

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 */
}
9. Mapping 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.

- 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 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.

- 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 NOTIFICATION-TYPE macro invocation is ignored.

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

```plaintext
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 {
        leaf ifIndex {
            type keyref {
                path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
            }
        }
    }
}
```
config false;
description
"Automagically generated keyref leaf.";
}
}

container linkDown-ifAdminStatus {
leaf ifIndex {
  type keyref {
    path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
  }
  config false;
description
  "Automagically generated keyref leaf.";
}
leaf ifAdminStatus {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
  }
  config false;
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 {
leaf ifIndex {
  type keyref {
    path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
  }
  config false;
description
  "Automagically generated keyref leaf.";
}
leaf ifOperStatus {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
  }

  "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).";
}
enum testing { value 3; }
enum unknown { value 4; }
enum dormant { value 5; }
enum notPresent { value 6; }
enum lowerLayerDown { value 7; }

cfg false;
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.";
10. IANA Considerations

TBD
11. Security Considerations

TBD
12. References

12.1. Normative References


12.2. Non-Normative References

Appendix A. Open Issues

- Need to add support for the translation of OBJECT-IDENTITY macro invocations to YANG identity statements.
- Investigate the translation of SMIv2 conformance statements into YANG features.
- Update the current SMIv2 NOTIFICATION-TYPE mapping to use YANG leafrefs.
- Spell out the regular expression generation algorithm (difficult).
Author’s Address

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de