This file is used to track issues that were discussed during the development of the SMIv2 to YANG translation in the IETF NETMOD working group. This issues covered here concern major design choices; this file does not attempt to track minor clarification requests etc.

To comment on issues on the mailing list, please include the issue number in the subject line of the email message.

* smi2yang-01: ambiguity of OID descriptors (container nesting) [closed]

  Producing containers of OID value assignments is difficult since multiple descriptors may refer to the same OID value:

```
foo OBJECT IDENTIFIER ::= { baz 1 }
bar OBJECT IDENTIFIER ::= { baz 1 }
```

  Simple rules like choosing the first defined descriptor do not work either since OID value assignments may be reordered in SMIv2 modules and new ones can be added.

** Solution #01-01:

- Drop the idea to represent the OID tree hierarchy by means of nested YANG containers. Instead, do the following:

  o Generate a top-level container named after the MODULE-IDENTITY node and put _all_ scalars into this container.

  o Translate all regular tables into top-level container named after the *Table node. The table container contains a list named after the *Entry node.

  o Translate all augmenting tables, that is the augmenting *Entry nodes, into top-level augment statements (dropping the *Table nodes).

- This solution results in a flat schema structure and it avoids using OID descriptors, thus the problem is solved.

- If OID tree information is desirable to be captured in the

generated YANG module, we could have extension statements to represent the OID structure, e.g.:

```
smiv2:oid 1.3.6.1.2.1.2 { smiv2:alias "interfaces"; }
smiv2:oid 1.3.6.1.2.1.2.1 { smiv2:alias "ifNumber"; }
```

This may in particular be useful since OBJECT-IDENTITY macros define constants that are used as values.

** Resolution

Adopted solution #01-01, but without changing the smiv2:oid statement, which needs further discussion. Opening smi2yang-08.

* smi2yang-02: update ambiguity of named INTEGERs and BITS [closed]

  SMIv2 allows the descriptors of enumerated INTEGERs and named BITS to change during revisions. YANG does not allow enums to be changed.

** Solution #02-01

   Declare this a non-problem since label changes are rare. Hm.

** Solution #02-02

   Do generate artificial labels that use the number and add the SMIv2 label via an smiv2 extension statement. This leads to rather ugly instance documents.

** Resolution

   RFC 4181 already says that labels MUST NOT change and provides a good explanation of the issue. Hence, it seems fair to adopt solution #02-01.

* smi2yang-03: OBJECT-IDENTITY containers [closed]

  OBJECT-IDENTITY macros are translated into YANG identities. They should in addition be translated into containers if the WG decides to keep containers, but note the ambiguities of this (smi2yang-01).

** Resolution

  By adopting #01-01, this became a mood point.

* smi2yang-04: string vs. binary [open]

It would be nice to have a reliable mechanism to determine whether an OCTET STRING can be represented as a human readable string or is a binary string. The current ID suggests to use the presence of a DISPLAY-HINT for this purpose. However, the SMIv2 allows to add a DISPLAY-HINT when a MIB module is revised, which can lead to different translations of different versions of a MIB module.

** Solution #04-01

Declare this a non-problem since additions of DISPLAY-HINTs are rare. Suggest that implementations provide options to cast an OCTET STRING into binary to handle situations where a DISPLAY-HINT got added when a module was revised.

** Solution #04-02

Translate strings into unions that allows both a human readable string representation and a binary representation. However, this requires to have some mechanism in place to decide whether a value is a human readable string or a binary string, which means we likely have to mess around with values.

** Solution #04-03

Always translate OCTET STRING types into the YANG binary types. We could list some well-known types as exceptions, like DisplayString, SnmpAdminString, DateAndTime but such a list will always be incomplete and hence some human readable data will be encoded into base64.

** Solution #04-04

Always translate OCTET STRINGS into a choice which contains a binary leaf. Add a second string leaf if there is a DISPLAY-HINT. This would also apply to objects using an OCTET STRING TC, that is in the extreme case (no special rules for well known string types), we get for sysDescr:

```
choice sysDescr-leaf {
  leaf sysDescr-string {
    type string;
  }
  leaf sysDescr-binary {
    type binary;
  }
}
```

While this solution is robust wrt. module revisions that add a DISPLAY-HINT, the cost of having to support two formats are significant.

* smi2yang-05: accessible-for-notify OBJECT-TYPES [closed]

  SMIv2 objects can have a MAX-ACCESS of accessible-for-notify and thus can’t be accessed via get operations.

** Solution #05-01

   Accessible-for-notify objects should be treated specially:

   - Do not generate schema tree leafs for accessible-for-notify objects.

   - If such objects are lised in the OBJECTS clause of a NOTIFICATION-TYPE, generate a proper leaf in the notification sub-statements instead of the typical leafref leaf.

** Resolution

   Adopted solution #05-01.

```
* smi2yang-06: notification containers [closed]

  For every object listed in the OBJECTS clause, a notification
  container is generated. This is necessary since multiple leafs may
  be needed to repesent an object since we have to capture any index
  information. Right now, the name of this container is:

    <notification-name>-<object-name>[-<number>]

   --n linkDown
    +--ro linkDown-ifIndex
    |  +--ro ifIndex?   leafref
    +--ro linkDown-ifAdminStatus
    |  +--ro ifIndex?         leafref
    |  +--ro ifAdminStatus?   leafref
    +--ro linkDown-ifOperStatus
       +--ro ifIndex?        leafref
       +--ro ifOperStatus?   leafref

  The choice of <notification-name>-<object-name> seems arbitrary
  and the optional disambiguating number may be a surprise.

** Solution #06-01:

   Always produce a number and call the container object-<number>,
   that is:

   --n linkDown
    +--ro object-1
    |  +--ro ifIndex?   leafref
    +--ro object-2
    |  +--ro ifIndex?         leafref
```

```
    |  +--ro ifAdminStatus?   leafref
    +--ro object-3
       +--ro ifIndex?        leafref
       +--ro ifOperStatus?   leafref
```

This seems to be closer to SMIv2 and is easier to implement as well.

** Resolution

Adopted solution #06-01.

* smi2yang-07: augments must be top-level schema tree nodes [closed]

The current I-D <draft-schoenw-netmod-smi-yang-02> shows the following translation of an augmenting table:

```
  container ifXTable {
    description
     "A list of interface entries.  The number of entries is
      given by the value of ifNumber.  This table contains
      additional objects for the interface table."

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

      // ...
    }
```

This is not legal YANG since the augment-stmt is only allowed as a top-level body statement in YANG.

** Solution #07-01

Move the augments to the top-level. We will loose the description of the table node. State that the table description MAY be translated into comments. Note that this is related to the resolution of smi2yang-01.

** Resolution

Adopted solution #07-01.

* smi2yang-08: inline vs. outside smiv2:oid mapping [closed]

The current translation uses the smiv2:oid to denote the OID of constructs that are mapped to YANG. However, this leaves out any constructs or OID descriptors that are not mapped to YANG.

** Solution #08-01

Do not inline OID definition but instead create a separate "section" at the end of a YANG module that just contains the OID mappings:

```
smiv2:oid 1.3.6.1.2.1.2 { smiv2:alias "interfaces"; }
smiv2:oid 1.3.6.1.2.1.2.1 { smiv2:alias "ifNumber"; }
```

This means less "noise" in the YANG definitions and it allows to capture all known OID descriptors within the YANG module. The downside is that in order to find out the OID for say a leaf like ifNumber, one has to do a lookup.

** Solution #08-02

Do inline OID definitions wherever possible, e.g.,

```
leaf ifNumber {
  //..
  smiv2:oid "1.3.6.1.2.1.2.1";
}
```

but allow smiv2:oid with nested smiv2:alias statements at other places to capture OIDs that do not map to YANG constructs.

** Solution #08-03

The smiv2:oid can be used inline (where it makes sense) as in

```
leaf ifNumber {
  //..
  smiv2:oid "1.3.6.1.2.1.2.1";
}
```

and in addition at the top-level (with nested smiv2:alias statements) as shown below:

```
smiv2:oid 1.3.6.1.2.1.2          { smiv2:alias interfaces; }
smiv2:oid 1.3.6.1.2.1.2.1        { smiv2:alias ifNumber; }
smiv2:oid 1.3.6.1.2.1.2.2        { smiv2:alias ifTable; }
smiv2:oid 1.3.6.1.2.1.2.2.1      { smiv2:alias ifEntry; }
smiv2:oid 1.3.6.1.2.1.2.2.1.1    { smiv2:alias ifIndex; }
smiv2:oid 1.3.6.1.2.1.2.2.1.2    { smiv2:alias ifDescr; }
```

For a translator, both are easy to generate and a user gets two reading benefits (and technically the OID now provides the link between the YANG leaf ifNumber and the SMIv2 OID descriptor ifNumber).

** Solution #08-04

The smiv2:oid can be used inline (where it makes sense) as in

```
leaf ifNumber {
  //..
  smiv2:oid "1.3.6.1.2.1.2.1";
}
```

and in addition at the top-level we allow smiv2:alias statements with embedded smiv2:oid statements as shown below:

```
smiv2:alias interfaces { smiv2:oid 1.3.6.1.2.1.2; }
smiv2:alias ifNumber   { smiv2:oid 1.3.6.1.2.1.2.1; }
smiv2:alias ifTable    { smiv2:oid 1.3.6.1.2.1.2.2; }
smiv2:alias ifEntry    { smiv2:oid 1.3.6.1.2.1.2.2.1; }
smiv2:alias ifDescr    { smiv2:oid 1.3.6.1.2.1.2.2.1.2; }
smiv2:alias ifIndex    { smiv2:oid 1.3.6.1.2.1.2.2.1.1; }
```

This is essentially solution #08-03 with the nesting of the smiv2:oid and smiv2:alias statements reversed. This solution does not require to change the semantics of smiv2:oid depending on whether it has substatements or not (the OID always "belongs" to the identifier defined by the "parent" statement).

** Resolution

Adopted solution #08-04.

* smi2yang-09: multiple top-level containers [closed]

  The current translation creates multiple top-level containers. This is somewhat at odds with a guideline given in RFC 6087:

  > There SHOULD only be one top-level data node defined in each YANG module, if any data nodes are defined at all.

  I do not recall the precise reasoning behind this SHOULD so it is not clear to me whether this is a problem or not.

** Solution #09-01

   Declare that the SHOULD allows to us to have multiple top-level containers.

** Solution #09-02

   Create a toplevel container (e.g. named after the module identity node) and register all scalars and non-augmenting tables below that node so we comply to the rule defined in RFC 6087.

** Resolution

   Adopted solution #09-02.

* smi2yang-10: "reference" substatement for "revision" statements [closed]

  RFC 6087 requires that every revision statement MUST have a reference statement. While this requirement makes sense for newly written modules, it does not make sense for modules derived from SMIv2 modules where this information is usually provided in the description clause.

** Solution #10-01

   Declare that this is really a bug in RFC 6087 - the requirement should have been a SHOULD not a MUST to allow for SMIv2 derived modules that do not provide an explicit reference.

** Resolution

   As Martin pointed out, this really only becomes an issue if the IETF ever publishes YANG modules generated from MIB modules as RFCs.

* smi2yang-11: toplevel node for SMIv1 modules [open]

  SMIv1 modules do not have a MODULE-IDENTITY node that can be used as the top-level node. The specification does not detail what happens in this case.

** Solution #11-01:

   Use the module name as the top-level node in such cases (this is what libsmi currently does) - but this has an issue if an SMIv1 module gets converted to SMIv2

** Solution #11-02:

   Use the shorter prefix generated from the module name - but this has the same issue as #11-01

** Solution #11-03:

   Always use the module name at the toplevel node - a bit more verbose but works the same in all cases, regardless whether an input module is in SMIv1 or SMIv2 format. Perhaps the name can be translated to lowercase (eye candy).

* smi2yang-12: smi:defval in SMI scope or YANG scope [open]

  Is the value reported in the smi:defval clause an SMI default value or a YANG default value (e.g. does it include prefixes)?

** Solution #12-01:

   It is the SMI default value; in particular OBJECT-IDENTITIES are showing up as SMIv2 descriptors and not as prefixed YANG identifiers.

* smi2yang-13: import rules cleanup [strawman: adopt #13-02]

  The current text on imports can be simplified and it can be less specific than it is right now (no need to special case snmpTraps for example).

** Solution #13-01:

   Describe a simple direct translation and state that implementations MAY prune imports that are not referenced in the translated YANG module. (The details of the pruning of unused imports is thus an implementation detail.)

   Question: Do we have to have to imports something that is only referenced in an smi:defval statement (assuming it is using SMI scope)?

** Solution #13-02

   David Reid wrote good replacement text, see thread "mib to yang conversion: imports".

* smi2yang-14: unflatten scalars [open]

With the resolution of #01, we ended with having all scalars registered right below the top-level node. This makes retrieving just those scalars (often collections of counters) a bit tricky because the naive approach will also retrieve the tables.

** Solution #14-01

Ignore this and assume people write proper filters.

** Solution #14-02

Move all scalars into an artificially created container, so we end up with:

```
foo-mib
  +-- scalars
  |     +-- fooScalar1
  |     +-- fooScalar2
  +-- fooTable1
  +-- fooTable2
```

** Solution #14-03

Use the name of the parent commonly used in the OID value assignment of the object type definition to group scalars, that is given

```
foo OBJECT IDENTIFIER ::= { something 1 }
bar OBJECT IDENTIFIER ::= { something 1 }
```

```
a OBJECT-TYPES --...-- ::= { foo 1 }
b OBJECT-TYPES --...-- ::= { bar 1 }
```

would give us

```
foo/a
bar/b
```

under the assumption that the OID value expression in the OBJECT-TYPE invocation never changes (which is something the SMIv2 update rules might allow) and we need to deal with uncommon but legal OID value definitions like

```
c OBJECT-TYPE --...-- ::= { something goo(1) 1 }
```

This has is related to smi2yang-01, but the details are slightly different.

* smi2yang-15: smiv2:implied [strawman: accept #15-01]

  Add an extension to indicate if an index object is IMPLIED. I
  believe that with this one addition a tool could get all of the MIB
  information it needs from the converted yang module.

** Solution #15-01

```
extension implied {
  argument "index";
  description
   "If an SMIv2 INDEX object is preceded by the IMPLIED keyword, then
    the implied statement is present in the yang module and takes as
    an argument the name of the IMPLIED index object.";
  reference
   "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}
```

* smi2yang-16: smiv2:oid oid representations [open]

  Modify the description clause of the oid extension to allow a more user-friendly format. For example, ifEntry could be written as ifTable.1 instead of 1.3.6.1.2.1.2.2.1, or perhaps just .1 as a short hand notation.

** Solution #16-01

```
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 dotted decimal notation. If an oid statement
    exists for the parent object identifier, then the value may
    be written in a short hand notation of the form
    <parent descriptor>.<last subidentifier> or just
    .<last subidentifier>. For example, the oid value for ifTable from
    IF-MIB would be '1.3.6.1.2.1.2.2'. The value for ifEntry could be
    '1.3.6.1.2.1.2.2.1', 'ifTable.1', or simply '.1'. Similarily,
    ifDescr could be '1.3.6.1.2.1.2.2.1.2', 'ifEntry.2', or '.2'";
  reference
   "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}
```

** Solution #16-02

  This seems mostly important for human writers not for machine translations. If more notations are needed, these should be separate aliases so as to allow easy processing of the OID value. Hence, no change needed wrt. smiv2:oid.

** Solution #16-03

```
extension subid {
  argument "value";
  description
   "The subid statement takes as an argument the most significant
    sub-identifier of the object identifier assigned to an SMIv2
    definition. The sub-identifier value is a single positive
    decimal whole number. The subid statement may not be used
    as a substatement to any top-level node in a YANG document.
    The subid substatement may be used only as a substatement to a
    node having a parent node defined with either a smiv2:oid or
    smiv2:subid substatement.";
  reference
   "RFC2578: Structure of Management Information Version 2 (SMIv2)";
}
```

* smi2yang-17: smiv2:* optional or mandatory? [strawman: accept #17-01]

  Should the generation of smiv2 statements be optional or part of the normal elements of procedure?

** Solution #17-01

   Make the generation of smiv2 statements part of the normal elements of procedure.

** Solution #17-02

   Keep the generation of smiv2 statements optional.

** Solution #17-03

   Make the generation of smiv2 statements a SHOULD.

* smi2yang-18: incomplete well known type mappings [strawman: accept #18-01]

  Extend the special type mappings as it seems rather incomplete.

** Solution #18-01

   Add explicit mapping for ZeroBasedCounter* types. They are listed as having an equivalent type in the YANG world in Table 1 of RFC 6021.

   Map TruthValue to the YANG boolean.

** Solution #18-02

   Remove all special case type mappings.

* smi2yang-19: accessible-for-notify objects in INDEXes [closed]

  If a table index is accessible-for-notify, it is not translated to yang. For example, in MIP-MIB (RFC 2006), the object mipSecViolatorAddress is a table index and has a MAX-ACCESS of accessible-for-notify. According to the current conversion rules (section 8.1), mipSecViolatorAddress will be dropped. But since it is a table index, it must be preserved.

** Solution #19-01

   Add the following sentence as the second sentence in section 1.8: Additionally, columnar objects with a MAX-ACCESS of accessible-for-notify are translated to YANG leaf statements if that columnar object is part of the INDEX clause of the table containing that columnar object.

** Resolution

   Adopted solution #19-01.

* smi2yang-20: duplicate objects in INDEXes [strawman: adopt #20-01]

  RMON2-MIB (RFC 4502) has an INDEX clause in which the same object appears twice. Specifically, alHostEntry's INDEX clause uses protocolDirLocalIndex twice. Following the current conversion rules results in a translated document that violates the YANG rule that a leaf identifier MUST NOT appear more than once in the key.

** Solution #20-01

  o The SMIv2 INDEX clause is mapped to the YANG key clause listing the columnar objects forming the key of the YANG list. If the same object appears more than once in the INDEX clause, append '_<n>' to the duplicate object names where '<n>' counts the occurances of the object in the INDEX clause, starting from 2.

  This would result in

```
key "hlHostControlIndex alHostTimeMark protocolDirLocalIndex "
    "nlHostAddress protocolDirLocalIndex_2";
```

* smi2yang-21: prefix generation [open]

The I-D proposes an algorithm to generate prefixes from module names.

** Solution #21-01

The current algorithm produces short readable prefixes in most of the cases. The algorithm is not perfect but prefixes are generally shorter than the module name and still readable/meaningful. The length has a great impact on leafref statements for example.

** Solution #21-02

I think the full name as a prefix makes the document more clear to the reader. In most cases, the algorithm either does not shorten the name or only shortens it slightly. Not using the prefix algorithm can make for some long path statements in a few cases. But I still think the clarity of the full name outweighs the brevity of the shortened names. There are a few cases in standard MIBs where using the prefix algorithm results in different modules using the same prefix to refer to different modules or different prefixes to refer to the same module. That’s perfectly legal yang, but I find it a bit confusing. The MIBs defined in RFC 5324 provide good examples that could support either side of this debate. That is, the prefix algorithm results in significantly shorter names, but could be confusing.