Using Smidump to Convert MIB to XSD
draft-li-mib-convert-00.txt

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of 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 December 22, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This memo depicts how the smidump tool converts a MIB to an XSD.
# Table of Contents

1. Introduction .................................................. 3
2. Mapping of the Module Definition .............................. 3
3. Mapping of the IMPORTS/FROM Clause ........................... 4
4. Mapping of the MODULE-IDENTITY Macro ....................... 4
   4.1. Mapping of the DESCRIPTION Clause ...................... 4
5. Mapping of the Managed Objects ............................... 5
   5.1. The Flattened Structure of Four Layers .................. 5
   5.2. Deriving of the Third-level Elements ..................... 7
   5.3. Mapping of the Leaf Objects ............................... 10
6. Mapping of the Datatypes ..................................... 12
7. Security Considerations ...................................... 14
8. IANA Considerations ......................................... 14
9. Acknowledgements ............................................. 14
10. References .................................................. 15
   10.1. Normative References ..................................... 15
   10.2. Informative References .................................. 15
Appendix A. The Smi.xsd File .................................... 15
1. Introduction

Accessing MIBs using NETCONF [ACC-MIB] describes a simple mechanism for accessing the Management Information Base (MIB), using the existing NETCONF [RFC4741] RPC infrastructure. It uses XSDs [ref.XMLSchema] generated by smidump [ref.libsmi] as data model. This memo depicts how the smidump tool converts a MIB to an XSD, and what the XSD looks like.

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].

2. Mapping of the Module Definition

The DEFINITIONS ::= BEGIN/END statement is used to start a MIB module definition in the following format:

```
<module-name> DEFINITIONS ::= BEGIN
...
END
```

where <module-name> is the name of the MIB module, for example, IF-MIB.

The DEFINITIONS statement is mapped to XSD as follows:

```
<?xml version="1.0"?>
<xsd:schema
  targetNamespace=  "http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/<module-name>"
  xmlns=        "http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/<module-name>"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:smi="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/smi"
  ...  
xml:lang="en"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  ...
</xsd:schema>
```
3. Mapping of the IMPORTS/FROM Clause

The IMPORTS/FROM statement is used to reference external objects from other MIB modules in the following format:

```
IMPORTS
    <descriptor> [, <descriptor> ... [, <descriptor>] ]
FROM <module-name>

[ ... ]
    <descriptor> [, <descriptor> ... [, <descriptor>] ]
FROM <module-name> ];
```

Each IMPORTS clause is mapped to two portions:
- A declaration of namespace attribute of the `<xsd:schema>` element:
  ```xml
  xmlns:<module-name>=
    "http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/<module-name>"
  ```
- A `<xsd:import>` element that imports the foregoing namespace:
  ```xml
  <xsd:import
    namespace=
    "http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/<module-name>"
    schemaLocation=
    "http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/<module-name>.xsd"/>
  ```

4. Mapping of the MODULE-IDENTITY Macro

Except for the DESCRIPTION clause of the MODULE-IDENTITY macro, the information contained in the MODULE-IDENTITY macro is discarded.

4.1. Mapping of the DESCRIPTION Clause

The DESCRIPTION clause contains a high-level textual description about the contents of this MIB module in the following format:

```
DESCRIPTION
    "..."
```

The DESCRIPTION clause is mapped to the `<xsd:documentation>` element in the `<xsd:annotation>` element, the XSD is as follows:

```
<xsd:annotation>
    <xsd:documentation>
        ...
    </xsd:documentation>
</xsd:annotation>
```
5. Mapping of the Managed Objects

5.1. The Flattened Structure of Four Layers

The smidump tool does not remain the hierarchy of the MIB tree. A "flattened" structure of the XML instance documents was invented. This flattened structure is divided into four layers, the top two levels are independent of specific MIBs, while the bottom two levels represent the concrete MIB objects.

The following figure illustrates the flattened structure of four layers.

```
<snmp-data>
  |<context>
  |  |<container of scalar elements (one or more)>
  |  |  |<scalar objects>
  |  |<An instance of table entry (one or more)>
  |  |  |<columnar objects>
```

1. The `<snmp-data>` element is the root element of an XML instance document. It may contain an arbitrary number of `<context>` elements.

2. The `<context>` element is located at the second level. It represents an agent context. It has six attributes:

   - `ipaddr`: Indicates the IP address of the agent context. The tuple of `ipaddr` and `port` and `community` attribute identifies a context. It must be present.
   - `hostname`: Indicates the hostname of the agent context. It need not be present.
   - `port`: Indicates the TCP/UDP port of the agent context. It must be present.
   - `community`: Indicates the community string of SNMPv1 and SNMPv2c. It must be present.
   - `caching`: This attribute is used for the implementation of an SNMP-to-XML gateway. It may indicate the expiry time of the cached data that was retrieved from an SNMP agent and that is requested by an XML client. It depends on specific implementation, thus it can be ignored. This attribute need not be present.
3. The third-level elements depend on specific MIBs. They can either represent containers of scalar elements that appear at most once, or instances of objects that are derived from table entries and thus can appear multiple times. Note that the list of these elements is not limited to a single MIB module. While scalar container elements do not have any attributes, the table entry elements include one or more index attributes to uniquely identify the instances. These attributes are derived from the table entry’s INDEX clause.

4. The fourth-level elements represent all leaf nodes of MIBs, i.e. scalar objects or columnar objects. There is no deeper level of element containment.

For the top two level, the XSD represents as follows:

```xml
<xsd:element name="snmp-data">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="context" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        ...
                    </xsd:sequence>
                    <xsd:attribute name="ipaddr" type="xsd:NMTOKEN" use="required"/>
                    <xsd:attribute name="hostname" type="xsd:NMTOKEN"/>
                    <xsd:attribute name="port" type="xsd:unsignedInt" use="required"/>
                    <xsd:attribute name="community" type="xsd:NMTOKEN" use="required"/>
                    <xsd:attribute name="caching" type="xsd:NMTOKEN"/>
                    <xsd:attribute name="time" type="xsd:dateTime" use="required"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:complexType>
```
5.2. Deriving of the Third-level Elements

There are two kinds of elements in the third level of the flattened structure.

1. The container element, which contains scalar elements and appears at most once. This kind of element is derived from an OBJECT IDENTIFIER assignment. The assignment usually appears in the following manner:

   <descriptor>    OBJECT IDENTIFIER
                   ::= { <parent-node> <sub-identifier> }

   The assignment is mapped to two portions. One is a complex type definition, which contains all scalar elements underneath the OID subtree; the other is an element declaration, which is named as <descriptor> and associates with the foregoing complex type. The XSD represents as follows:

   <xsd:element name="<descriptor>" type="<descriptor>Type"
            minOccurs="0">
     <xsd:annotation>
       <xsd:appinfo>
         <maxAccess>unknown</maxAccess>
         <oid>...</oid>
         <status>unknown</status>
       </xsd:appinfo>
     </xsd:annotation>
   </xsd:element>

   ...

   <xsd:complexType name="<descriptor>Type">
     <xsd:sequence>
        <!-- declarations of scalar element -->
     </xsd:sequence>
   </xsd:complexType>

2. The entry instance element, which contains columnar elements and can appear multiple times. This kind of element is derived from the OBJECT-TYPE macro that defines a table entry. The definition usually appears in the following manner:
<descriptor> OBJECT-TYPE
SYNTAX  ...
MAX-ACCESS not-accessible
STATUS  "current"|"deprecated"|"obsolete"
DESCRIPTION
  "...
INDEX   { <index1>, ... }
::= { <parent-node> 1 }
The OBJECT-TYPE macro is mapped to a complex type definition and an element declaration. The macro-descriptor in OBJECT-TYPE MACRO is mapped as the name of the element. The SYNTAX clause is mapped as the ‘type’ attribute of the element declaration. The MAX-ACCESS and STATUS clauses are mapped as the <maxAccess> and <status> in the <appinfo> of the element declaration respectively. The OBJECT-TYPE Value is calculated and mapped as the value of the <oid> in the <appinfo> of the element declaration. The DESCRIPTION clause is mapped as <documentation> of the element declaration. Each object in the INDEX clause is mapped as an attribute declaration of the complex type. The XSD represents as follows:
<xsd:element name="<descriptor>" type="<descriptor>Type" minOccurs="0" maxOccurs="unbounded">
<xsd:annotation>
  <xsd:appinfo>
    <flags create="yes"/>
    <maxAccess>not-accessible</maxAccess>
    <oid>...</oid>
    <status>...</status>
  </xsd:appinfo>
  <xsd:documentation>...
</xsd:documentation>
</xsd:annotation>
</xsd:element>
...
<xsd:complexType name="<descriptor>Type">
  <xsd:sequence>
    <!-- declarations of columnar element -->
    <xsd:attribute name="<index1>" type="..." use="required">
      <xsd:annotation>
        <xsd:appinfo>
          <maxAccess>...</maxAccess>
          <oid>...</oid>
          <status>...</status>
        </xsd:appinfo>
        <xsd:documentation>...
      </xsd:documentation>
    </xsd:annotation>
  </xsd:sequence>
</xsd:complexType>

If the table entry can be created, a <flags> element will be present in the <appinfo> of the element declaration.

5.3. Mapping of the Leaf Objects

Both scalar and columnar objects are defined by the OBJECT-TYPE macro, thus their mappings are the same. The definition of leaf object usually appears in the following manner:
<descriptor> OBJECT-TYPE
SYNTAX ... [UNITS ...]
MAX-ACCESS "not-accessible" | "read-only"
| "read-write" | "read-create"
STATUS "current" | "deprecated" | "obsolete"
DESCRIPTION "..."
[REFERENCE "..."]
[DEFVAL ...]
::= { <parent-node> <sub-identifier> }

Where the UNITS, REFERENCE and DEFVAL clauses are optional.

Each OBJECT-TYPE macro is mapped to an element declaration. The macro-descriptor in OBJECT-TYPE macro is mapped as the element name. The SYNTAX clause is mapped as the 'type' attribute of the element declaration or an anonymous simple type definition. The REFERENCE clause is discarded. Other clauses are mapped as elements in the <annotation> of the element declaration. The mapping relationship between clauses and elements is illustrated as the following table:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-ACCESS</td>
<td>&lt;maxAccess&gt;</td>
</tr>
<tr>
<td>UNITS</td>
<td>&lt;units&gt;</td>
</tr>
<tr>
<td>STATUS</td>
<td>&lt;status&gt;</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>&lt;documentation&gt;</td>
</tr>
<tr>
<td>DEFVAL</td>
<td>&lt;default&gt;</td>
</tr>
</tbody>
</table>

The XSD represents as follows:

```
<xsd:element name="<descriptor>" type="..." minOccurs="0">
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess>...</maxAccess>
      <oid>...</oid>
      <status>...</status>
      <units>...</units>
    </xsd:appinfo>
    <xsd:documentation>...
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```
6. Mapping of the Datatypes

Smidump provides an XSD file named smi.xsd, which selects some primitive types from SMI and maps them to the XSD build-in datatypes or its derived datatypes. Other types, even the types defined in the SMIPv2, are derived from these primitive types, such as Counter32, TimeTicks. The mapping is showed as bellow. See the XML schema in Appendix A for more detail.

<table>
<thead>
<tr>
<th>SMI primitive type</th>
<th>XSD Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer32</td>
<td>int</td>
</tr>
<tr>
<td>OctetString</td>
<td>hexBinary</td>
</tr>
<tr>
<td>ObjectIdentifier</td>
<td>string</td>
</tr>
<tr>
<td>Unsigned32</td>
<td>unsignedInt</td>
</tr>
<tr>
<td>Unsigned64</td>
<td>unsignedLong</td>
</tr>
</tbody>
</table>

There are many textual conventions in MIBs. A textual convention is a sub-type of a base type with restriction on size, range or enumerations, furthermore, it conveys a more precise semantics. It is used by the SYNTAX clause of the OBJECT-TYPE macro, in the same way as a base type is used. The TEXTUAL-CONVENTION macro is used to define a textual convention. The usage is showed as below:

```xml
<descriptor> ::= TEXTUAL-CONVENTION
                  [DISPLAY-HINT "..."]
                 STATUS "current"|"deprecated"|"obsolete"
                 DESCRIPTION "...
                 [REFERENCE "..."]
                 SYNTAX <base-type> [<restriction>]
```

Where the DISPLAY-HINT and REFERENCE clauses are optional.

A textual convention is mapped as a simple type definition with corresponding base type and restriction. The STATUS and REFERENCE clauses are discarded. The descriptor in TEXTUAL-CONVENTION macro is mapped as the simple type name. The DESCRIPTION clause is mapped as the <documentation>; the DISPLAY-HINT clause is mapped as the <appinfo>; the SYNTAX clause is mapped as the <restriction>. The mapping is showed as bellow:
Besides the TEXTUAL-CONVENTION macro can define a refinement of a base type, the SYNTAX clause of the OBJECT-TYPE macro can define a local refinement of a base type also. In this case, the refinement is mapped as an anonymous simple type definition, and the element declaration does not use the 'type' attribute to indicate the datatype of the element.

Smidump usually maps SMI base type and its refinement to corresponding XSD datatypes and restrictions. But there is a few exceptions:

Enumerated INTEGER: Smidump maps an enumerated INTEGER as a NMTOKEN datatype with restriction on enumeration. The label for the number becomes the enumerated value. The number is mapped as the value of the <intVal> element, which is contained in the <appinfo> for the enumerated value. The XSD represents as follows:

```xml
<xsd:simpleType>
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="xxx">
      <xsd:annotation>
        <xsd:appinfo>
          <intVal>1</intVal>
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="yyy">
      <xsd:annotation>
        <xsd:appinfo>
          <intVal>2</intVal>
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:enumeration>
    ...
  </xsd:restriction>
</xsd:simpleType>
```
The BITS construct: Smidump maps a BITS construct as a list with restriction on maximum length, the maximum length of the list is determined by the number of bit. Each list item is an enumerated NMTOKEN. The XSD represents as follows:

```xml
<xsd:simpleType>
  <xsd:restriction>
    <xsd:list>
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="xxx">
            <xsd:annotation>
              <xsd:appinfo>
                <intVal>0</intVal>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:enumeration>
          ...<xsd:enumeration value="zzz">
            <xsd:annotation>
              <xsd:appinfo>
                <intVal>n-1</intVal>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:enumeration>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:list>
  </xsd:restriction>
  <xsd:maxLength value="n"/>
</xsd:simpleType>
```

7. Security Considerations

TBD.

8. IANA Considerations

TBD.

9. Acknowledgements

This document is based on the libsmi implementation and the corresponding papers. Therefore the author would like to acknowledge
the developers of the libsmi toolkit. In particular, Thanks to Frank Strauss, Torsten Klie and Juergen Schoenwaelder.

10. References

10.1. Normative References

[RFC2119] Bradner, s., "Key words for RFCs to Indicate Requirements Levels", RFC 2119, March 1997.


10.2. Informative References


Appendix A. The Smi.xsd File

<xsd:schema
  targetNamespace="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/smi"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/smi">  
  <xsd:simpleType name="Integer32">
    <xsd:restriction base="xsd:int"/>
  </xsd:simpleType>
  
  <xsd:simpleType name="OctetString">
    <xsd:restriction base="xsd:hexBinary">
      <xsd:maxLength value="65535" fixed="false"/>
    </xsd:restriction>
  </xsd:simpleType>
  
  <xsd:simpleType name="ObjectIdentifier">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="2"/>
      <xsd:maxLength value="128"/>
      <xsd:pattern value="([0-2]\.|[0-9]+)*/">
    </xsd:restriction>
  </xsd:simpleType>
  
  <xsd:simpleType name="Unsigned32">
    <xsd:restriction base="xsd:unsignedInt"/>
  </xsd:simpleType>
  
  <xsd:simpleType name="Unsigned64">
    <xsd:restriction base="xsd:unsignedLong"/>
  </xsd:simpleType>
  
  <xsd:simpleType name="EncAttrType">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="ascii"/>
      <xsd:enumeration value="hex"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
Author’s Address

Yan Li
Huawei Technologies
No.3 Xinxi Road, Shangdi Information Industry Base
Beijing, HaiDian District  100085
P.R.China

Phone: +86 10 8288 2008
EMail: liyan_77@huawei.com