Conversion of MIB to XSD for NETCONF
draft-xiao-conversion-dm-00

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 September 27, 2008.
Abstract

NETCONF needs a data model for its process of standardization. This documentation defines a standard expression of SMI MIBs in XSD for NETCONF to ensure uniformity, general interoperability and reusability of existing MIBs. In addition, we define a XML schema to give a restriction and validation to translated XSD files.

Table of Contents

1. Introduction .............................................. 3
2. Conventions .............................................. 6
3. Requirements ............................................ 7
   3.1. Requirements for NETCONF .......................... 7
   3.2. Requirements for MIB ............................... 7
4. Mapping of data types .................................. 8
   4.1. SMI base datatypes ................................. 8
   4.2. Other datatypes .................................... 9
5. Mapping of MIB structure ............................... 11
6. Mapping and application of Marco clauses .............. 13
7. A XML schema for translated XSD ......................... 15
8. Security Considerations ............................... 16
9. IANA Considerations ................................... 17
10. References ............................................. 18
    10.1. Normative References ........................... 18
    10.2. Informative References ......................... 18
Appendix A. SMI-Data types.xsd .............................. 19
Appendix B. A SMI.xsd for NETCONF ......................... 25
Authors’ Addresses .......................................... 29
Intellectual Property and Copyright Statements .............. 30
1. Introduction

[NETCONF] can be conceptually partitioned into four layers:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>Configuration data</td>
</tr>
<tr>
<td>Operations</td>
<td>&lt;get-config&gt;, &lt;edit-config&gt;</td>
</tr>
<tr>
<td>RPC</td>
<td>&lt;rpc&gt;, &lt;rpc-reply&gt;</td>
</tr>
<tr>
<td>Transport</td>
<td>BEEP, SSH, SSL, console</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
</tbody>
</table>

The last three layers of NETCONF have been already standardized in RFC4741, RFC4742, RFC4743 and RFC4744. However, there isn’t a standard data modeling language or a standard data model for NETCONF content layer. If we can’t make the content layer of NETCONF standardized, every vendor can define its own data model, which will cause trouble and confusion in understanding the syntax and semantics of data model in communication. Thus the NETCONF won’t be applied widely as SNMP in future and the NETCONF defined in RFC4741 will have no sense.

The work to standardize the content layer of NETCONF is of two ways:

1. Create a new data modeling language and then a new data model for NETCONF. YANG is a new data modeling language which defines a new SMI for NETCONF containing datatypes, node statement, and syntax specification and so on. The NCX is somewhat like YANG. It not only defines the new SMI for NETCOFN but also has supplemented some ability to NETCONF protocol. All these new languages are under discussion, which means that these will be a longer term effort to create a solid SMI and then remodel some of the key data to be carried.

2. Conversion from MIB to XSD. This is being done by XSDMI group. The XSDMI effort is designed to produce a XSD specification by translating from MIB. NETCONF configuration is an improvement of CLI, not SNMP which has been widely used for performance,
monitoring and fault management. However, some MIB-based monitoring data have become part of the operational framework of many networks. And many of the data names and meanings have been widely accepted by vendors for years.

For a long run, to establish a new data modeling language and new data model is much better than simple conversion of MIB to XSD. However, its standardization will need a very long time and plenty of effort. On the other hand, IETF has spent over 20 years to make SMI MIB standardization and many vendors also have made great effort to supplement these MIBs which have been widely used in current network management systems. Most of these MIB data are focused on monitoring information, such as statistics and state. Although the NETCONF protocol are aiming at establishing a new data model for configuration management, it still need to benefit from reusing existing MIB objects for monitoring the configured technology or checking the state following configuration. It will be a waste to abandon these MIB modules without adequate reasons. So in current times, to convert SMI MIB module to XSD is more feasible than creating a new data model and reusing MIB for NETCONF content layer can be a transitional way before new language and new data model emerge.

NETCONF uses XML-based data encoding for the configuration data as well as the protocol messages. Under such background, we should provide a standard translation to make using the MIB’s managed objects with XSD easier. The whys and wherefores that XSD is considered a better way to be the data modeling language for NETCONF are as follows:

1. Data models which expressed by XSD can be accessed by NETCONF and any XML-based protocols such as IDMEF, XCAP, IDMEF, and ATOM, which improves the interoperability.

2. XSD can generate any diverse data types, multi-dimensional arrays and can be used in real world devices which employ hash-tables which don’t exist in SMI.

3. Many people are familiar with XML and XSD. There are many standard technologies about XML and will useful for protocol development.

The XSDMI BOF proposed the creation of a WG to develop:

a. the XSD equivalents of datatypes and textual conventions from SMIv2.
b. algorithms to translate MIB module specifications into XSD equivalents.

c. Netconf operations to access MIB data.

d. a draft documenting security requirements for protocols accessing the MIB.

Based on the XSDMI’s and previous smidump’s work, this documentation defines a standard expression of SMI MIBs in XSD for NETCONF to ensure uniformity and general interoperability and reusability of existing MIBs. In addition, we define a XML schema to give a restriction and validation to translated XSD files.
2. Conventions

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 [RFC2119].

Sections requiring further editing are identified by [TODO] markers in the text. Points requiring further WG research and discussion are identified by [DISCUSS] markers in the text.
3. Requirements

This section describes some basic restrictions on translated XSD from MIBs.

3.1. Requirements for NETCONF

NETCONF has an obvious advantage of its separation of configuration and state data. Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state. State data is the additional data on a system that is not configuration data such as read-only status information and collected statistics. First advantage lies in its separation of configuration data and state data, which can avoid the problems that comparisons of configuration data sets would be dominated by irrelevant entries such as different statistics and incoming data could contain nonsensical requests such as attempts write read-only data.

Our target is to establish a data model translated from MIB to XSD for NETCONF, so we should follow the requirements of NETCONF and need a separation of configuration and state data. Although MIBs are mostly used for monitor and don’t have explicit separation of them in any form of label, we can only use <MAX-ACCESS> label to distinguish them. We consider data whose <MAX-ACCESS> value are "not-accessible", "read-only" as state data’s and whose <MAX-ACCESS> value are "read-write", "read-create" as configuration.

[TODO] More requirements need to be standardized for NETCONF content, even a configuration network management.

3.2. Requirements for MIB

Two goals of this work are instrumentation reuse and semantics reuse. The IETF has spent twenty years developing standard managed objects, and vendors have supplemented that with proprietary managed objects, all written using a standard way expressing the syntax and semantics. So it is better to preserve that work and make it available for XML-based messaging, which means that we should follow the syntax and semantics rule of SMI.
4. Mapping of data types

4.1. SMI base datatypes

This section defines the IETF standard expression of SMI base datatypes in XSD. [I-D.li-mib-convert] has given an expression for SMI base datatypes, and there are few shortcomings of it:

1. It isn’t covering all datatypes of SMI base types, which can’t be incompatible with all different kind of vendors’ device.

2. Some of the datatypes’ names are different with SMI base datatypes, which can’t form a semantic merge of different devices.

In this section, the datatypes defined in RFC2578 and RFC2579 should be all translated in derived data types in XSD that the number and names of data types are the same to SMI. The data types are on the base of restriction on build-in data types in XSD. It not only preserves the unambiguous of data types but also reduces the changes that vendors make to be suitable for translated data types. Then the translated data types are written as a fixed file in order that if these data types are used in some module, the source module can directly import the only one file enough.

There are totally 24 kind of datatypes included in RFC2578 and RFC2579 except that some datatypes have been obsoleted such as "Opaque" and "InstancePointer". We can divide these datatypes into five groups by the way they derived from base types in XSD:

1. Some types can be equally placed by the build-in datatypes in XSD. What the difference between translated SMI datatypes and XSD datatypes is only the name of them. For example, Integer, Unsigned32, Counter32, Counter64, Gauge32, TimeTicks, OCTET STRING are of this type.

2. Some use pattern restriction on the base types to express translated datatypes which may not be unique such as IPAddress, MacAddress, PhyAddress, DateAndTime, Objectidentifier.

3. Some use enumeration way to express datatypes. Such as "StorageType", "RowStatus", "TruthValue".

4. Some use range restriction to express them such as "TAddress", "DisplayString", "TestAndIncr", "TimeInterval".

5. Some derives from defined datatypes such as "TimeStamp", "AutonomousType", "VariablePointer", "RowPointer", "Tdomain".
The full definition of datatypes’ schema is shown in Appendix A.

[Discuss] Is there any need to express datatypes of OID and those derived from OID? NETCONF-based network management needn’t OID to locate the managed objects. Instead, they use "Subtree" or "XPATH" to find them.

[Discuss] How we make the expression of all datatypes standard and unique for datatypes that use pattern restriction, for example, the regular expression of IPAddress isn’t unique, then how we deal with such thing?

[TODO] We should tell the protocol-independent datatypes from those protocol-special datatypes. NETCONF content is specified to protocol-independent thus its datatypes should also follow the rule.

4.2. Other datatypes

Other data types are mostly defined by vendors for their special use and are all in form of Textual Conventions. An example is described as follows:

```
OwnerString ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "255a"
    STATUS    deprecated
    DESCRIPTION
        "This data type is used to model an administratively assigned name of the owner of a resource."
    SYNTAX     OCTET STRING (SIZE(0..255))
```

The SYNTAX clause defines abstract data structure corresponding to the textual convention. We only use <restriction> to restrict the base type.

The translated XSD is representing as follows:
<xsd:simpleType name="OwnerString">
  <xsd:annotation>
    <xsd:appinfo>
      <displayHint>255a</displayHint>
    </xsd:appinfo>
    <xsd:documentation>
      This data type is used to model an administratively assigned name of the owner of a resource.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="OCTET STRING">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="255"/>
  </xsd:restriction>
</xsd:simpleType>

[Discuss] How we define a standard for the expression of restriction in TCs for auto conversion? For example, sometimes we can use "minInclusive" and "maxInclusive" to express the range of values, use "minlength" and "maxlength" to express the range of length, and use "pattern" to define regular expression.
5. Mapping of MIB structure

This section gives a flattened structure of translated XSD files from
SMI MIBs.

Previous MIB tree has so many layers that many of them are of little
use and some of the middle node are of no use only for the
organization of its children. Until now, there are many tools used
to converting MIBs, and one of the popular tools is smidump. Instead
of the high hierarchy of MIB tree, it uses a flattened structure
which only has four levels: the root of the document level, the agent
information description level, the third level representing either
the containers of scalar nodes or the entry of table nodes, and the
fourth level of all leaf nodes including scalars nodes or columnar
nodes. It omits many middle nodes and the conformance statement.
However, it also has following limits when used in NETCONF content
layer:

1. It can’t satisfy the requirements of NETCONF protocol that the
   configuration and state data should be separated.

2. The second layer is not needed in NETCONF protocol.

In our draft, we also optimize the complex structure of MIB to a
flattened and simple structure especially for NETCONF and its
separation of configuration and state data. The new structure still
reserves the relationship between leaf nodes and their parents as
following four layers:

1. The root of the document level.

2. Three branches of the root named "configuration", "state" and
   "notification".

3. The third level representing either the containers of scalar
   nodes or the entry of table nodes or the notification nodes.

4. The fourth level of all leaf nodes including scalars nodes or
   columnar nodes.

The XSD for the top two levels is represented as follows:
<xsd:element name="root">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="configuration" >
                <xsd:complexType>
                    <xsd:sequence>
                        ......
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:element>

<xsd:element name="state" >
    <xsd:complexType>
        <xsd:sequence>
            ......
        </xsd:complexType>
    </xsd:element>
</xsd:element>

<xsd:element name="configuration" >
    <xsd:complexType>
        <xsd:sequence>
            ......
        </xsd:complexType>
    </xsd:element>
</xsd:element>

<xsd:element name="notification" >
    <xsd:complexType>
        <xsd:sequence>
            ......
        </xsd:complexType>
    </xsd:element>
</xsd:element>
6. Mapping and application of Marco clauses

There are three kinds of elements in the third level depicted as follows:

1. The container element containing scalar elements which appear at most once. This kind of element is derived from an OBJECT IDENTIFIER or OBJECT IDENTIFIER assignment, of which the latter is the supplement of the former.

2. The entry of a table containing columnar elements which can appear multiple times. It is defined from the OBJECT-TYPE macro.

3. The notification node is used while the agent is off working defined by NOTIFICATION-TYPE macro.

Both scalar and columnar objects are defined by the OBJECT-TYPE macro as the leaf nodes, thus their mappings are the same.

The translations of OBJECT-TYPE, OBJECT IDENTIFIER, and NOTIFICATION-TYPE are more or less alike. The items included in their definition in SMI are translated into following items in XSD by the rule that the NETCONF manager only need to know the nodes’ name not all information about them such as "description" while communicate with NETCOFN agent.

We translate all the other information as <annotation> in XSD whose children is <appinfo> and <documentation>. The detailed mapping is shown as below.

<table>
<thead>
<tr>
<th>SMI primitive item</th>
<th>XSD item</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-ACCESS</td>
<td>&lt;maxAccess&gt;</td>
<td>OBJECT-TYPE</td>
</tr>
<tr>
<td>STATUS</td>
<td>&lt;status&gt;</td>
<td>OBJECT-TYPE, OBJECT IDENTIFIER</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>&lt;documentation&gt;</td>
<td>OBJECT-TYPE, OBJECT IDENTIFIER, NOTIFICATION-TYPE</td>
</tr>
<tr>
<td>INDEX</td>
<td>&lt;index&gt;</td>
<td>OBJECT-TYPE</td>
</tr>
<tr>
<td>OBJECTS</td>
<td>&lt;objects&gt;</td>
<td>NOTIFICATION-TYPE</td>
</tr>
</tbody>
</table>
The following is an example:

```plaintext
sysName OBJECT-TYPE
SYNTAX  DisplayString (SIZE (0..255))
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"An administratively-assigned name for this managed node. By convention, this is the node’s fully-qualified domain name."
::= { system 5 }
```

Each OBJECT-TYPE macro is mapped to an element declaration. Note that the UNITS, REFERENCE and DefValPart items are all omitted in translated XSD, and the SYNTAX, MAX-ACCESS and STATUS are all translated as the child of the `<appinfo>` while the DESCRIPTION are under `<documentation>` which is the child of `<annotation>` with `<appinfo>`.

The XSD represents as follows:

```xml
<xsd:element name="sysName" type="ccnu:DisplayString">
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess>read-write</maxAccess>
      <oid>1.3.6.1.2.1.1.5</oid>
      <status>mandatory</status>
    </xsd:appinfo>
    <xsd:documentation>
      An administratively-assigned name for this managed node. By convention, this is the node’s fully-qualified domain name.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

The example above shows the translation of leaf nodes. If there is a middle node, we should only modify the type as "ComplexType", and we define leaf node under "ComplexType".
7. A XML schema for translated XSD

Until now, there are many ways of translating MIB to XSD with different structure or content type, so there should be a schema to restrict them and make them follow a rule, when the rule changes, the translated XSD should also change.

The XML schema for translated XSD is as Appendix B.

From Appendix B, we can see that Many elements are defined as global elements so that they can be referred many times, for example, <annotation>, <appinfo>, <documentation>, <status> and so on. It reduces the waste of repeat definition and makes them more readable and easily understood.
8. Security Considerations

None.
9. IANA Considerations

None.
10. References

10.1. Normative References


10.2. Informative References


Appendix A. SMI-Data types.xsd

Converted from SMI core datatypes
</xsd:documentation></xsd:annotation>


<xsd:simpleType name="INTEGER">
  <xsd:annotation>
    <xsd:documentation>
      INTEGER from RFC 2578, page 8 and sec. 7.1.1.
      An enumerated integer is simply the ‘enum’ data type
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction xsd:base="xsd:int"/>
</xsd:simpleType>

<xsd:simpleType name="OCTET STRING">
  <xsd:annotation>
    <xsd:documentation>
      OCTET STRING from RFC 2578
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction xsd:base="xsd:string">
    <xsd:pattern value="([0-9a-zA-Z]{8}){1,255}"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="IpAddress">
  <xsd:annotation>
    <xsd:documentation>
      IpAddress from RFC 2578
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction xsd:base="xsd:string"/>
  <xsd:pattern value="(((0-9)[1-2][0-9-9]|0-9)[0-9-9]|2[0-4][0-9]|25[0-5]):
     (0-9 [1,2] [1-09-9][0-9-9]|2 [0-4][0-9-9]|25 [0-5])"/>
</xsd:simpleType>
<xsd:simpleType name="Counter32">
    <xsd:annotation>
        <xsd:documentation>Counter32 from RFC 2578</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedInt"/>
</xsd:simpleType>

<xsd:simpleType name="Gauge32">
    <xsd:annotation>
        <xsd:documentation>Gauge32 from RFC 2578</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedInt"/>
</xsd:simpleType>

<xsd:simpleType name="Unsigned32">
    <xsd:annotation>
        <xsd:documentation>Unsigned32 from RFC 2578</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedInt"/>
</xsd:simpleType>

<xsd:simpleType name="TimeTicks">
    <xsd:annotation>
        <xsd:documentation>TimeTicks from RFC 2578</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedInt"/>
</xsd:simpleType>

<xsd:simpleType name="Counter64">
    <xsd:annotation>
        <xsd:documentation>Counter64 from RFC 2578</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedLong"/>
</xsd:simpleType>

<xsd:simpleType name="Unsigned64">
    <xsd:annotation>
        <xsd:documentation>Unsigned64 TC (missing) from RFC 2856.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:unsignedLong"/>
</xsd:simpleType>

<xsd:simpleType name="PhysAddress">
    <xsd:annotation>
        <xsd:documentation>PhysAddress TC from RFC 2579
    </xsd:annotation>
</xsd:simpleType>
<xsd:documentation>MacAddress TC from RFC 2579</xsd:documentation>
</xsd:simpleType>

<xsd:simpleType name="TruthValue">
    <xsd:annotation>
        <xsd:documentation>TruthValue TC from RFC 2579</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:string">
        <xsd:enumeration value="true"/>
        <xsd:enumeration value="false"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="DateAndTime">
    <xsd:annotation>
        <xsd:documentation>DateAndTime TC from RFC 2579</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction xsd:base="xsd:string">
        <xsd:pattern value="((0|1)[2|3|4|5|6](0-9){4})-(1[0-9]|0\d)-(0-9)\ ((2[0-3]|1[0-9]|0\d):([0-5]\d|\d)\ ((0-5)[0-9]|0\d)\ ((0-9)\ ((0-9)|1[0-9]|1\d):((0|1)[2|3|4|5](0-9))*/"></xsd:pattern>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TAddress">
    <xsd:annotation>
        <xsd:documentation>TAddress TC from RFC 2579</xsd:documentation>
    </xsd:annotation>
</xsd:simpleType>
<xsd:restriction xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xsd:base="OCTET STRING">
  <xsd:minLength value="1"/>
  <xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="DisplayString">
  <xsd:annotation>
    <xsd:documentation>
      DisplayString TC from RFC 2579
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="0"/>
    <xsd:maxLength value="255"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TestAndIncr">
  <xsd:annotation>
    <xsd:documentation>
      TestAndIncr TC from RFC 2579
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:unsignedInt">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="2147483647"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="RowStatus">
  <xsd:annotation>
    <xsd:documentation>
      RowStatus TC from RFC 2579
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="active"/>
    <xsd:enumeration value="notInService"/>
    <xsd:enumeration value="notReady"/>
    <xsd:enumeration value="createAndGo"/>
    <xsd:enumeration value="createAndWait"/>
    <xsd:enumeration value="destroy"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="StorageType">
  <xsd:annotation>

Xiao & Chang       Expires September 27, 2008       [Page 22]
<xsd:documentation>
StorageType TC from RFC 2579
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="other"/>
<xsd:enumeration value="volatile"/>
<xsd:enumeration value="nonVolatile"/>
<xsd:enumeration value="permanent"/>
<xsd:enumeration value="readOnly"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TimeStamp">
<xsd:annotation>
<xsd:documentation>
TimeStamp TC from RFC 2579
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="TimeTicks"/>
</xsd:simpleType>

<xsd:simpleType name="TimeInterval">
<xsd:annotation>
<xsd:documentation>
TimeInterval TC from RFC 2579
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:unsignedInt">
<xsd:minInclusive value="0"/>
<xsd:maxInclusive value="2147483647"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ObjectIdentifier">
<xsd:annotation>
<xsd:documentation>
OBJECT IDENTIFIER from RFC 2578, libsmi v0.4.5 output.
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xs:string">
<xsd:pattern value="[0-2](\.(0|([1-9][0-9]*)))*/"/>
<xsd:minLength value="2"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="AutonomousType">
</xsd:complexType>

Xiao & Chang Expires September 27, 2008 [Page 23]
<xs:documentation>
  AutonomousType TC from RFC 2579, page 5.
</xs:documentation>
<xs:simpleContent>
  <xs:extension base="ObjectIdentifier"/>
</xs:simpleContent>
</xs:complexType>

<xs:complexType name="RowPointer">
  <xs:annotation>
    <xs:documentation>
      VariablePointer TC from RFC 2579
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="ObjectIdentifier"/>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="VariablePointer">
  <xs:annotation>
    <xs:documentation>
      smidump v0.4.5,
      A pointer to a specific object instance.
      For example, sysContact.0 or ifInOctets.3.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="ObjectIdentifier"/>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="TDomain">
  <xs:annotation>
    <xs:documentation>
      TDomain TC from RFC 2579, page 20.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="ObjectIdentifier"/>
  </xs:simpleContent>
</xs:complexType>

</xsd:schema>
Appendix B. A SMI.xsd for NETCONF

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetnamespace= "http://netcom/smitoxsd/smischema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
schemaLocation="http://www.w3.org/2001/xml.xsd"/>

<xsd:element name="element" type="roottype">
  <xsd:attribute name="name" use="required" default="root"/>
</xsd:element>

<xsd:complexType name="roottype">
  <xsd:element name="complexType">
    <xsd:complexType>
      <xsd:element name="sequence" type="sequencetype"/>
    </xsd:complexType>
  </xsd:element>
</xsd:complexType>

<xsd:element name="sequence" type="sequenceType"/>

<xsd:complexType name="sequenceType">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="element" type="elementnotificationtype">
        <xsd:attribute name="name" use="required" default="notification"/>
      </xsd:element>
      <xsd:element name="element" type="elementconfigtype">
        <xsd:attribute name="name" use="required" default="config"/>
      </xsd:element>
      <xsd:element name="element" type="elementstatetype">
        <xsd:attribute name="name" use="required" default="state"/>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexType>

<xsd:complexType name="elementnotificationtype">
  <xsd:sequence>
    <xsd:element name="complexType">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="element">
            <xsd:complexType>
              <xsd:element ref="annotation"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:complexType>
</xsd:complexType>
</xsd:complexType>
  <xsd:attribute name="name" type="xsd:string"
    use="required"/>
  <xsd:attribute name="maxOccurs" use="required"
    default="unbounded"/>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:complexType>
<xsd:complexType name="elementconfigtype">
  <xsd:element name="complexType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="element" type="elementleaftype">
          <xsd:complexType>
            <xsd:element ref="annotation"/>
          </xsd:complexType>
          <xsd:attribute name="name" type="xsd:string"
            use="required"/>
          <xsd:attribute name="maxOccurs" use="required"
            default="unbounded"/>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:complexType>
</xsd:complexType>
<xsd:complexType name="elementstatetype">
  <xsd:element name="complexType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="element" type="elementleaftype">
          <xsd:complexType>
            <xsd:element ref="annotation"/>
          </xsd:complexType>
          <xsd:attribute name="name" type="xsd:string"
            use="required"/>
          <xsd:attribute name="maxOccurs" use="required"
            default="unbounded"/>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:complexType>
</xsd:complexType>
</xsd:complexType>
</xsd:complexType>
</xsd:complexType>
<xsd:complexType name="elementleaftype">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="element">
        <xsd:complexType>
          <xsd:element ref="annotation"/>
        </xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="maxOccurs" use="required" default="unbounded"/>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexType>

<xsd:element name="annotation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="documentation"/>
      <xsd:element ref="appinfo"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="appinfo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="status" minOccurs="0"/>
      <xsd:element ref="maxAccess" minOccurs="0"/>
      <xsd:element ref="oid"/>
      <xsd:element ref="objects" minOccurs="0"/>
      <xsd:element ref="status" minOccurs="0"/>
      <xsd:element ref="units" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="documentation">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="maxAccess">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="accessible-for-notify"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
<xsd:element name="status">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="deprecated"/>
      <xsd:enumeration value="current"/>
      <xsd:enumeration value="obsolete"/>
      <xsd:enumeration value="unknown"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="oid" type="xsd:string"/>

</xsd:schema>
Authors’ Addresses

Debao Xiao
Institute of Computer Network and Communication
Huazhong Normal University
Wuhan, Hubei 430079
P.R.China

Phone: +86 027 6786 6108
Email: dbxiao@mail.ccnu.edu.cn

Yanan Chang
Institute of Computer Network and Communication
Huazhong Normal University
Wuhan, Hubei 430079
P.R.China

Phone: +86 027 6786 6108
Email: cyn_23@yahoo.com.cn