Conversion of MIB to XSD for NETCONF
draft-xiao-conversion-dm-02

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 March 15, 2010.

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 in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
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 ................... 12
   6.1. the mapping of MODULE-IDENTITY macro .................. 12
   6.2. the mapping of OBJECT-IDENTIFIER macro ................ 14
   6.3. the mapping of OBJECT-TYPE macro ................... 14
     6.3.1. the mapping of scalar nodes and columnar nodes .... 14
     6.3.2. the mapping of entry nodes .......................... 16
   6.4. the mapping of NOTIFICATION macro ................... 18
7. A XML schema for translated XSD .............................. 21
8. Security Considerations ...................................... 22
9. IANA Considerations .......................................... 23
10. References .................................................... 24
   10.1. Normative References .................................... 24
   10.2. Informative References .................................. 24
Appendix A. SMI-Data types.xsd .................................. 25
Appendix B. A SMI.xsd for NETCONF ................................ 38
Authors’ Addresses ............................................. 43
1. Introduction

[NETCONF] can be conceptually partitioned into four layers:

+------------+-----------------------------+
|    Layer   |           Example           |
+------------+-----------------------------+
|   Content  |      Configuration data     |
|            |                             |
| Operations | <get-config>, <edit-config> |
|            |                             |
|     RPC    |      <rpc>, <rpc-reply>     |
|            |                             |
|  Transport |        BEEP, SSH, TLS       |
+------------+-----------------------------+

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 NETCONF 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 "write-only", "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:annotation>
</xsd:simpleType>
This data type is used to model an administratively assigned name of the owner of a resource.

[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.
6. Mapping and application of Marco clauses

In the flattened structure of four levels, it includes nodes which are defined by MODULE-IDENTITY macro, OBJECT-IDENTIFIER macro, OBJECT-TYPE macro and NOTIFICATION-TYPE macro.

We translate all the macro types as follows:

6.1. the mapping of MODULE-IDENTITY macro

Some mibs have MODULE-IDENTITY macro, which is used to describe the entire block of information for top-level description of the mib. We translate MODULE-IDENTITY macro as information of root element. Note that the LAST-UPDATED, ORGANIZATION![cents]CONTACT-INFO![cents]REVISION and the corresponding label elements DESCRIPTION are all omitted in the translated XSD, and the DESCRIPTION which is the introduction of the mib is translated as <documentation>, which is the child of <annotation>.

The following is an example, which is from IF-MIB:

ifMIB MODULE-IDENTITY

    LAST-UPDATED "200006140000Z"

    ORGANIZATION "IETF Interfaces MIB Working Group"

    CONTACT-INFO

        " 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 "200006140000Z"

DESCRIPTION
"Clarifications agreed upon by the Interfaces MIB WG, and
published as RFC 2863."

REVISION "199602282155Z"

DESCRIPTION
"Revisions made by the Interfaces MIB WG, and published in
RFC 2233."

REVISION "199311082155Z"

DESCRIPTION
"Initial revision, published as part of RFC 1573."

::= { mib-2 31 }

The XSD represents as follows:
<xsd:element name="IF-MIB" type="IF-MIBtype">
  <xsd:annotation>
    <xsd:documentation>
      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.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
6.2. the mapping of OBJECT-IDENTIFIER macro

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. The information of OBJECT-IDENTIFIER macro contained is relatively simple. We translate OID as <oid>, which is placed under <appinfo>, as a child of <annotation>.

The following is an example, which is from mib of RFC1213:

```
system OBJECT IDENTIFIER ::= { mib-2 1 }
```

The XSD represents as follows:

```
<xsd:element name="system" type="systemType">
  <xsd:annotation>
    <xsd:appinfo>
      <oid>.1.3.6.1.2.1.1</oid>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

6.3. the mapping of OBJECT-TYPE macro

In the bottom of the flatted structure, there is scalar nodes and the columnar nodes which are defined in macro OBJECT-TYPE. And entry nodes are also defined in macro OBJECT-TYPE.

6.3.1. the mapping of scalar nodes and columnar nodes

For this type of nodes, UnitPart, ReferPart, IndexPart, DefValPart are all omitted in translated XSD. We translate MAX-ACCESS, STATUS as <maxAccess> and <status>, which are placed under <appinfo>, and <DESCRIPTION> is mapping as <documentation>.

The following is an example, which is from mib of RFC1213:

```
sysObjectID OBJECT-TYPE
```
SYNTAX OBJECT IDENTIFIER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The vendor’s authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining ‘what kind of box’ is being managed. For example, if vendor ‘Flintstones, Inc.’ was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its ‘Fred Router’.
::= { system 2 }

The XSD represents as follows:
<xsd:element name="sysObjectID" type="ccnu:ObjectIdentifier">
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess>read-only</maxAccess>
      <status>mandatory</status>
    </xsd:appinfo>
    <xsd:documentation>
      The vendor’s authoritative identification of the
network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining ‘what kind of box’ is being managed. For example, if vendor ‘Flintstones, Inc.’ was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its ‘Fred Router’.

</xsd:documentation>
</xsd:annotation>
</xsd:element>

6.3.2. the mapping of entry nodes

The translations of entry nodes are more or less alike. But there are two differences: 1) entry nodes can appear many times. So, we add two attributes <minOccurs> and <maxOccurs> to the translated XSD. 2) we need to use index clause to read entry node, we add <index> element to mark it. But if it is an external index, add <augments> instead.

The following is an example, which is from mib of RFC1213:

ifEntry OBJECT-TYPE

SYNTAX IfEntry

ACCESS not-accessible

STATUS mandatory

DESCRIPTION

"An interface entry containing objects at the subnetwork layer and below for a particular
interface."

INDEX { ifIndex }
::= { ifTable 1 }

The XSD represents as follows:

```xml
<xsd:element name="ifEntry" type="ccnu:ObjectIdentifier">
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess>not-accessible</maxAccess>
      <status>mandatory</status>
    </xsd:appinfo>
    <index>ifIndex</index>
    <xsd:documentation>
      An interface entry containing objects at the
      subnetwork layer and below for a particular
      interface.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

The following is an example from mib of IF-MIB, which is an entry node contain an external index:

```py
ifXEntry OBJECT-TYPE
  SYNTAX IfXEntry
  MAX-ACCESS not-accessible
  STATUS current
  DESCRIPTION
```
"An entry containing additional management information applicable to a particular interface."

AUGMENTS { ifEntry }

::= { ifXTable 1 }

The XSD represents as follows:

<xsd:element name="ifXEntry" type="ccnu:ObjectIdentifier">
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess>not-accessible</maxAccess>
      <status>current</status>
    </xsd:appinfo>
    <augments>ifIndex</augments>
    <xsd:documentation>
      An entry containing additional management information applicable to a particular interface.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

6.4. the mapping of NOTIFICATION macro

The notification node is used while the agent is off working defined by NOTIFICATION-TYPE macro. ReferPart is omitted in translated xsd, STATUS, OBJECTS are translated as <status> </status> and <objects>, which are placed under <appinfo>, as a child of <annotation>. And DESCRIPTION is translated as <documentation>, which is placed under <annotation>.

The following is an example, which is from IF-MIB:

linkDown NOTIFICATION-TYPE
OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }

STATUS current

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."

::= { snmpTraps 3 }

The XSD represents as follows:

<xsd:element name="linkDown">
  <xsd:annotation>
    <xsd:appinfo>
      <status>current</status>
      <objects>
        ifIndex, ifAdminStatus, ifOperStatus
      </objects>
    </xsd:appinfo>
    <xsd:documentation>
      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.
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

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://netcom/smitoxsdd/smidatatypes"
targetNamespace="http://netcom/smitoxsdd/smidatatypes"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
      Converted from SMI core datatypes
    </xsd:documentation>
  </xsd:annotation>

schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <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:schema>
<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}|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.(0-9){1,2}|1[0-9][0-9]|0-9){3}\.(0-9){1,2}|1[0-9][0-9]|0-9)|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:simpleType>
<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>
    </xsd:documentation>
  </xsd:annotation>
</xsd:simpleType>
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:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(\[0-9A-Fa-f]{2}\:)*([0-9A-Fa-f]{2})"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="MacAddress">
  
  <xsd:annotation>
    <xsd:documentation>
      MacAddress TC from RFC 2579.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
  </xsd:restriction>
</xsd:simpleType>
<xsd:pattern value="([0-9A-Fa-f]{2}:){5}([0-9A-Fa-f]{2})"/>
</xsd:restriction>
</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:restriction>
</xsd:simpleType>
<xsd:pattern value=" ((0|1|2|3|4|5|6)(0-9){4})\- \\
(11|12)(0-9)\- (30|31| [1-2][0-9])\, \(2[0-3]\ |
1[0-9] | [0-9])\:(0-5)[0-9])|0-9)\:(0-5) [0-9]| [0-9]):[0-9])\, (\+|\-)([0-9]|10|11)\:
((0|1|2|3|4|5)[0-9])"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TAddress">
  <xsd:annotation>
    <xsd:documentation>
      TAddress TC from RFC 2579
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction 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:simpleType>
<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:simpleType>
<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>
    <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 xsd: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:annotation>
    <xsd:documentation>
      AutonomousType TC from RFC 2579, page 5.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="ObjectIdentifier"/>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="RowPointer"/>
VariablePointer TC from RFC 2579.

VariablePointer TC from RFC 2579, page 6. smidump v0.4.5, A pointer to a specific object instance. For example, sysContact.0 or ifInOctets.3.
<xsd:annotation>
  <xsd:documentation>
    TDomain TC from RFC 2579, page 20.
  </xsd:documentation>
</xsd:annotation>

<xsd:simpleContent>
  <xsd:extension base="ObjectIdentifier"/>
</xsd:simpleContent>

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

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

  <xsd:element name="root" type="roottype">
    </xsd:element>

  <xsd:complexType name="roottype">
    <xsd:sequence>
      <xsd:element name="config" type="configtype"/>
      <xsd:element name="status" type="statustype"/>
      <xsd:element name="notification" type="notificationtype"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="configtype">
    <xsd:sequence>
      <xsd:element name="element1" type="element1type">
        <xsd:annotation>
          <xsd:appinfo>
            <oid/>
          </xsd:appinfo>
        </xsd:annotation>
      </xsd:element1type>
    </xsd:sequence>
  </xsd:complexType>
</xsd:appinfo>
</xsd:annotation>
</xsd:element>

<xsd:element name="element2" type="element2type" minOccurs="0" maxOccurs="bounded">
  
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess/>
      <oid/>
      <status/>
    </xsd:appinfo>
  </xsd:annotation>

  <xsd:documentation/>

</xsd:element>

...

</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="statustype">
  
  <xsd:sequence>
    <xsd:element name="element1" type="element1type">
      
      <xsd:annotation>
        <xsd:appinfo>
          <oid>
            ...
          </oid>
        </xsd:appinfo>
      </xsd:annotation>

    </xsd:element>

  </xsd:sequence>

</xsd:complexType>
</xsd:annotation>
</xsd:element>

<xsd:element name="element2" type="syntax" minOccurs="0" maxOccurs="bounded">
  
  <xsd:annotation>
    <xsd:appinfo>
      <maxAccess/>
      <oid/>
      <status/>
    </xsd:appinfo>
    <xsd:documentation/>
  </xsd:annotation>

  </xsd:element>

...
<xsd:complexType name="elementType">
  <xsd:sequence>
    <xsd:element name="element1" type="syntax">
      <xsd:annotation>
        <xsd:appinfo>
          <maxAccess/>
          <oid/>
        </xsd:appinfo>
        <xsd:documentation/>
      </xsd:annotation>
    </xsd:element>
    ...<br>
    <xsd:element name="elementN">
      <xsd:annotation>
        <xsd:appinfo>
          <maxAccess/>
          <oid/>
        </xsd:appinfo>
        <xsd:documentation/>
      </xsd:annotation>
    </xsd:element>
    ...
  </xsd:sequence>
</xsd:complexType>
<xsd:appinfo>
  <maxAccess/>
  <oid/>
  <status/>
</xsd:appinfo>
<xsd:documentation/>
</xsd:annotation>
<xsd:simpleType>
  <xsd:restriction base="syntax"/>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="element">
  <xsd:restriction/>
</xsd:simpleType>

</xsd:schema>
Authors’ Addresses

Xiaoqiong Wu
Institute of Computer Network and Communication
Huazhong Normal University
Wuhan, HuBei 430079
P.R. China

Phone: +86 027 6136 8682
Email: marissblue@yahoo.com.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

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

Phone: +86 027 6136 8682
Email: dbxiao@mail.ccnu.edu.cn