Network Working Group   J. Schoenwaelder, Ed.
Internet-Draft   Jacobs University
Intended status: Standards Track   May 13, 2009
Expires: November 14, 2009

Common YANG Data Types
draft-ietf-netmod-yang-types-03

Status of this Memo

Copyright Notice

Abstract

This document introduces a collection of common data types to be used with the YANG data modeling language.

Table of Contents

1. Introduction

YANG [YANG] is a data modeling language used to model configuration and state data manipulated by the NETCONF [RFC4741] protocol. The YANG language supports a small set of built-in data types and provides mechanisms to derive other types from the built-in types.

This document introduces a collection of common data types derived from the built-in YANG data types. The definitions are organized in several YANG modules. The "ietf-yang-types" module contains generally useful data types. The "ietf-inet-types" module contains definitions that are relevant for the Internet protocol suite.

The derived types are generally designed to be applicable for modeling all areas of management information.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

2. Overview

This section provides a short overview over the types defined in subsequent sections and their equivalent SMIv2 data types. Table 1 list the types defined in the ietf-yang-types YANG module and the corresponding SMIv2 types (if any).

ietf-yang-types

Yang type	Equivalent SMIv2 type (module)
counter32	Counter32 (SNMPv2-SMI)
zero-based-counter32	ZeroBasedCounter32 (RMON2-MIB)
counter64	Counter64 (SNMPv2-SMI)
zero-based-counter64	ZeroBasedCounter64 (HCNUM-TC)
gauge32	Gauge32 (SNMPv2-SMI)
gauge64	CounterBasedGauge64 (HCNUM-TC)
object-identifier	-
object-identifier-128	OBJECT IDENTIFIER
date-and-time	-
timeticks	TimeTicks (SNMPv2-SMI)
timestamp	TimeStamp (SNMPv2-TC)
phys-address	PhysAddress (SNMPv2-TC)
mac-address	MacAddress (SNMPv2-TC)
xpath1.0	-

Table 1

Table 2 list the types defined in the ietf-inet-types YANG module and the corresponding SMIv2 types (if any).

inet-yang-types

Yang type	Equivalent SMIv2 type (module)
ip-version	-
dscp	Dscp (DIFFSERV-DSCP-TC)
ipv6-flow-label	IPv6FlowLabel (IPV6-FLOW-LABEL-MIB)
port-number	InetPortNumber (INET-ADDRESS-MIB)
as-number	InetAutonomousSystemNumber (INET-ADDRESS-MIB)
ip-address	-
ipv4-address	-
ipv6-address	-
ip-prefix	-
ipv4-prefix	-
ipv6-prefix	-
domain-name	-
host	-
uri	Uri (URI-TC-MIB)

Table 2

3. Core YANG Derived Types

```
 module ietf-yang-types {

   namespace "urn:ietf:params:xml:ns:yang:yang-types";
   prefix "yang";

   organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

   contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: David Partain
               <mailto:david.partain@ericsson.com>

     WG Chair: David Kessens
               <mailto: david.kessens@nsn.com>

     Editor:   Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>";

   description
    "This module contains a collection of generally useful derived
     YANG data types.

     Copyright (c) 2009 IETF Trust and the persons identified as
     the document authors.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, are permitted provided that the
     following conditions are met:

     - Redistributions of source code must retain the above
       copyright notice, this list of conditions and the
       following disclaimer.

     - Redistributions in binary form must reproduce the above
       copyright notice, this list of conditions and the
       following disclaimer in the documentation and/or other
       materials provided with the distribution.

     - Neither the name of Internet Society, IETF or IETF
       Trust, nor the names of specific contributors, may be
       used to endorse or promote products derived from this
       software without specific prior written permission.
```

```
     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
     CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED
     WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
     PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
     OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.

     This version of this YANG module is part of RFC XXXX; see
     the RFC itself for full legal notices.";
   // RFC Ed.: replace XXXX with actual RFC number and remove this note

   revision 2009-05-13 {
     description
      "Initial revision, published as RFC XXXX.";
   }
   // RFC Ed.: replace XXXX with actual RFC number and remove this note

   /*** collection of counter and gauge types ***/

   typedef counter32 {
     type uint32;
     description
      "The counter32 type represents a non-negative integer
       which monotonically increases until it reaches a
       maximum value of 2^32-1 (4294967295 decimal), when it
       wraps around and starts increasing again from zero.

       Counters have no defined `initial' value, and thus, a
       single value of a counter has (in general) no information
       content.  Discontinuities in the monotonically increasing
       value normally occur at re-initialization of the
       management system, and at other times as specified in the
       description of an object instance using this type.  If
       such other times can occur, for example, the creation of
       an object instance of type counter32 at times other than
       re-initialization, then a corresponding object should be
       defined, with an appropriate type, to indicate the last
       discontinuity.

       The counter32 type should not be used for configuration
```

```
       objects. A default statement should not be used for
       attributes with a type value of counter32.

       This type is in the value set and its semantics equivalent
       to the Counter32 type of the SMIv2.";
     reference
      "RFC 2578: Structure of Management Information Version 2 (SMIv2)";
   }

   typedef zero-based-counter32 {
     type yang:counter32;
     default "0";
     description
      "The zero-based-counter32 type represents a counter32
       which has the defined `initial' value zero.

       Objects of this type will be set to zero(0) on creation
       and will thereafter count appropriate events, wrapping
       back to zero(0) when the value 2^32 is reached.

       Provided that an application discovers the new object within
       the minimum time to wrap it can use the initial value as a
       delta since it last polled the table of which this object is
       part.  It is important for a management station to be aware
       of this minimum time and the actual time between polls, and
       to discard data if the actual time is too long or there is
       no defined minimum time.

       This type is in the value set and its semantics equivalent
       to the ZeroBasedCounter32 textual convention of the SMIv2.";
     reference
       "RFC 2021: Remote Network Monitoring Management Information
                  Base Version 2 using SMIv2";
   }

   typedef counter64 {
     type uint64;
     description
      "The counter64 type represents a non-negative integer
       which monotonically increases until it reaches a
       maximum value of 2^64-1 (18446744073709551615), when
       it wraps around and starts increasing again from zero.

       Counters have no defined `initial' value, and thus, a
       single value of a counter has (in general) no information
       content.  Discontinuities in the monotonically increasing
       value normally occur at re-initialization of the
       management system, and at other times as specified in the
```

```
       description of an object instance using this type.  If
       such other times can occur, for example, the creation of
       an object instance of type counter64 at times other than
       re-initialization, then a corresponding object should be
       defined, with an appropriate type, to indicate the last
       discontinuity.

       The counter64 type should not be used for configuration
       objects. A default statement should not be used for
       attributes with a type value of counter64.

       This type is in the value set and its semantics equivalent
       to the Counter64 type of the SMIv2.";
     reference
      "RFC 2578: Structure of Management Information Version 2 (SMIv2)";
   }

   typedef zero-based-counter64 {
     type yang:counter64;
     default "0";
     description
      "The zero-based-counter64 type represents a counter64 which
       has the defined `initial' value zero.

       Objects of this type will be set to zero(0) on creation
       and will thereafter count appropriate events, wrapping
       back to zero(0) when the value 2^64 is reached.

       Provided that an application discovers the new object within
       the minimum time to wrap it can use the initial value as a
       delta since it last polled the table of which this object is
       part.  It is important for a management station to be aware
       of this minimum time and the actual time between polls, and
       to discard data if the actual time is too long or there is
       no defined minimum time.

       This type is in the value set and its semantics equivalent
       to the ZeroBasedCounter64 textual convention of the SMIv2.";
     reference
      "RFC 2856: Textual Conventions for Additional High Capacity
                 Data Types";
   }

   typedef gauge32 {
     type uint32;
     description
      "The gauge32 type represents a non-negative integer, which
       may increase or decrease, but shall never exceed a maximum
```

```
         value, nor fall below a minimum value.  The maximum value
         can not be greater than 2^32-1 (4294967295 decimal), and
         the minimum value can not be smaller than 0.  The value of
         a gauge32 has its maximum value whenever the information
         being modeled is greater than or equal to its maximum
         value, and has its minimum value whenever the information
         being modeled is smaller than or equal to its minimum value.
         If the information being modeled subsequently decreases
         below (increases above) the maximum (minimum) value, the
         gauge32 also decreases (increases).

         This type is in the value set and its semantics equivalent
         to the Counter32 type of the SMIv2.";
       reference
        "RFC 2578: Structure of Management Information Version 2 (SMIv2)";
     }

     typedef gauge64 {
       type uint64;
       description
        "The gauge64 type represents a non-negative integer, which
         may increase or decrease, but shall never exceed a maximum
         value, nor fall below a minimum value.  The maximum value
         can not be greater than 2^64-1 (18446744073709551615), and
         the minimum value can not be smaller than 0.  The value of
         a gauge64 has its maximum value whenever the information
         being modeled is greater than or equal to its maximum
         value, and has its minimum value whenever the information
         being modeled is smaller than or equal to its minimum value.
         If the information being modeled subsequently decreases
         below (increases above) the maximum (minimum) value, the
         gauge64 also decreases (increases).

         This type is in the value set and its semantics equivalent
         to the CounterBasedGauge64 SMIv2 textual convention defined
         in RFC 2856";
       reference
        "RFC 2856: Textual Conventions for Additional High Capacity
                   Data Types";
     }

     /*** collection of identifier related types ***/

     typedef object-identifier {
       type string {
         pattern '(([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*))))'
               + '(\.(0|([1-9]\d*)))*';
       }
```

```
     description
      "The object-identifier type represents administratively
       assigned names in a registration-hierarchical-name tree.

       Values of this type are denoted as a sequence of numerical
       non-negative sub-identifier values. Each sub-identifier
       value MUST NOT exceed 2^32-1 (4294967295). Sub-identifiers
       are separated by single dots and without any intermediate
       white space.

       Although the number of sub-identifiers is not limited,
       module designers should realize that there may be
       implementations that stick with the SMIv2 limit of 128
       sub-identifiers.

       This type is a superset of the SMIv2 OBJECT IDENTIFIER type
       since it is not restricted to 128 sub-identifiers.";
     reference
      "ISO/IEC 9834-1: Information technology -- Open Systems
       Interconnection -- Procedures for the operation of OSI
       Registration Authorities: General procedures and top
       arcs of the ASN.1 Object Identifier tree";
   }

   typedef object-identifier-128 {
     type object-identifier {
       pattern '\d*(.\d*){1,127}';
     }
     description
      "This type represents object-identifiers restricted to 128
       sub-identifiers.

       This type is in the value set and its semantics equivalent
       to the OBJECT IDENTIFIER type of the SMIv2.";
     reference
      "RFC 2578: Structure of Management Information Version 2 (SMIv2)";
   }

   /*** collection of date and time related types ***/

   typedef date-and-time {
     type string {
       pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?'
             + '(Z|(\+|-)\d{2}:\d{2})';
     }
     description
      'The date-and-time type is a profile of the ISO 8601
       standard for representation of dates and times using the
```

```
       Gregorian calendar. The format is most easily described
       using the following ABFN (see RFC 3339):

       date-fullyear   = 4DIGIT
       date-month      = 2DIGIT  ; 01-12
       date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31
       time-hour       = 2DIGIT  ; 00-23
       time-minute     = 2DIGIT  ; 00-59
       time-second     = 2DIGIT  ; 00-58, 00-59, 00-60
       time-secfrac    = "." 1*DIGIT
       time-numoffset  = ("+" / "-") time-hour ":" time-minute
       time-offset     = "Z" / time-numoffset

       partial-time    = time-hour ":" time-minute ":" time-second
                         [time-secfrac]
       full-date       = date-fullyear "-" date-month "-" date-mday
       full-time       = partial-time time-offset

       date-time       = full-date "T" full-time

       The date-and-time type is consistent with the semantics defined
       in RFC 3339. The data-and-time type is compatible with the
       dateTime XML schema type with the following two notable
       exceptions:

       (a) The data-and-time type does not allow negative years.

       (b) The data-and-time time-offset -00:00 indicates an unknown
           time zone (see RFC 3339) while -00:00 and +00:00 and Z all
           represent the same time zone in dateTime.

       This type is not equivalent to the DateAndTime textual
       convention of the SMIv2 since RFC 3339 uses a different
       separator between full-date and full-time and provides
       higher resolution of time-secfrac.

       The canonical format for date-and-time values mandates the UTC
       time format with the time-offset is indicated by the letter "Z".
       This is consistent with the canonical format used by the
       dateTime XML schema type.';
     reference
      "RFC 3339: Date and Time on the Internet: Timestamps
       RFC 2579: Textual Conventions for SMIv2
       W3C REC-xmlschema-2-20041028: XML Schema Part 2: Datatypes
                 Second Edition";
   }

   typedef timeticks {
```

```
     type uint32;
     description
      "The timeticks type represents a non-negative integer which
       represents the time, modulo 2^32 (4294967296 decimal), in
       hundredths of a second between two epochs. When objects
       are defined which use this type, the description of the
       object identifies both of the reference epochs.

       This type is in the value set and its semantics equivalent
       to the TimeTicks type of the SMIv2.";
     reference
      "RFC 2578: Structure of Management Information Version 2 (SMIv2)";
   }

   typedef timestamp {
     type yang:timeticks;
     description
      "The timestamp type represents the value of an associated
       timeticks object at which a specific occurrence happened.
       The specific occurrence must be defined in the description
       of any object defined using this type.  When the specific
       occurrence occurred prior to the last time the associated
       timeticks attribute was zero, then the timestamp value is
       zero.  Note that this requires all timestamp values to be
       reset to zero when the value of the associated timeticks
       attribute reaches 497+ days and wraps around to zero.

       The associated timeticks object must be specified
       in the description of any object using this type.

       This type is in the value set and its semantics equivalent
       to the TimeStamp textual convention of the SMIv2.";
     reference
      "RFC 2579: Textual Conventions for SMIv2";
   }

   /*** collection of generic address types ***/

   typedef phys-address {
     type string {
       pattern '([0-9a0-fA-F]{2}(:[0-9a0-fA-F]{2})*)?';
     }
     description
      "Represents media- or physical-level addresses represented
       as a sequence octets, each octet represented by two hexadecimal
       numbers. Octets are separated by colons.

       This type is in the value set and its semantics equivalent
```

```
       to the PhysAddress textual convention of the SMIv2.";
     reference
      "RFC 2579: Textual Conventions for SMIv2";
   }

   typedef mac-address {
     type string {
       pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}';
     }
     description
      "The mac-address type represents an 802 MAC address represented
       in the `canonical' order defined by IEEE 802.1a, i.e., as if it
       were transmitted least significant bit first, even though 802.5
       (in contrast to other 802.x protocols) requires MAC addresses
       to be transmitted most significant bit first.

       This type is in the value set and its semantics equivalent to
       the MacAddress textual convention of the SMIv2.";
     reference
       "RFC 2579: Textual Conventions for SMIv2";
   }

   /*** collection of XML specific types ***/

   typedef xpath1.0 {
     type string;
     description
      "This type represents an XPATH 1.0 expression.";
     reference
      "W3C REC-xpath-19991116: XML Path Language (XPath) Version 1.0";
   }

 }
```

4. Internet Specific Derived Types

```
 module ietf-inet-types {

   namespace "urn:ietf:params:xml:ns:yang:inet-types";
   prefix "inet";

   organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

   contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: David Partain
               <mailto:david.partain@ericsson.com>

     WG Chair: David Kessens
               <mailto:david.kessens@nsn.com>

     Editor:   Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>";

   description
    "This module contains a collection of generally useful derived
     YANG data types for Internet addresses and related things.

     Copyright (c) 2009 IETF Trust and the persons identified as
     the document authors.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, are permitted provided that the
     following conditions are met:

     - Redistributions of source code must retain the above
       copyright notice, this list of conditions and the
       following disclaimer.

     - Redistributions in binary form must reproduce the above
       copyright notice, this list of conditions and the
       following disclaimer in the documentation and/or other
       materials provided with the distribution.

     - Neither the name of Internet Society, IETF or IETF
       Trust, nor the names of specific contributors, may be
       used to endorse or promote products derived from this
       software without specific prior written permission.
```

```
       THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED
       WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
       WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
       PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
       OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
       INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
       (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
       GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
       BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
       LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
       (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
       OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
       POSSIBILITY OF SUCH DAMAGE.

       This version of this YANG module is part of RFC XXXX; see
       the RFC itself for full legal notices.";
     // RFC Ed.: replace XXXX with actual RFC number and remove this note

     revision 2009-05-13 {
       description
        "Initial revision, published as RFC XXXX.";
     }
     // RFC Ed.: replace XXXX with actual RFC number and remove this note

     /*** collection of protocol field related types ***/

     typedef ip-version {
       type enumeration {
         enum unknown {
           value "0";
           description
            "An unknown or unspecified version of the Internet protocol.";
         }
         enum ipv4 {
           value "1";
           description
            "The IPv4 protocol as defined in RFC 791.";
         }
         enum ipv6 {
           value "2";
           description
            "The IPv6 protocol as defined in RFC 2460.";
         }
       }
       description
        "This value represents the version of the IP protocol.
```

```
       This type is in the value set and its semantics equivalent
       to the InetVersion textual convention of the SMIv2. However,
       the lexical appearance is different from the InetVersion
       textual convention.";
     reference
      "RFC  791: Internet Protocol
       RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
       RFC 4001: Textual Conventions for Internet Network Addresses";
   }

   typedef dscp {
     type uint8 {
       range "0..63";
     }
     description
      "The dscp type represents a Differentiated Services Code-Point
       that may be used for marking packets in a traffic stream.

       This type is in the value set and its semantics equivalent
       to the Dscp textual convention of the SMIv2.";
     reference
      "RFC 3289: Management Information Base for the Differentiated
                 Services Architecture
       RFC 2474: Definition of the Differentiated Services Field
                 (DS Field) in the IPv4 and IPv6 Headers
       RFC 2780: IANA Allocation Guidelines For Values In
                 the Internet Protocol and Related Headers";
   }

   typedef ipv6-flow-label {
     type uint32 {
       range "0..1048575";
     }
     description
      "The flow-label type represents flow identifier or Flow Label
       in an IPv6 packet header that may be used to discriminate
       traffic flows.

       This type is in the value set and its semantics equivalent
       to the IPv6FlowLabel textual convention of the SMIv2.";
     reference
      "RFC 3595: Textual Conventions for IPv6 Flow Label
       RFC 2460: Internet Protocol, Version 6 (IPv6) Specification";
   }

   typedef port-number {
     type uint16 {
       range "1..65535";
```

```
     }
     description
      "The port-number type represents a 16-bit port number of an
       Internet transport layer protocol such as UDP, TCP, DCCP or
       SCTP. Port numbers are assigned by IANA.  A current list of
       all assignments is available from <http://www.iana.org/>.

       Note that the value zero is not a valid port number. A union
       type might be used in situations where the value zero is
       meaningful.

       This type is in the value set and its semantics equivalent
       to the InetPortNumber textual convention of the SMIv2.";
     reference
      "RFC  768: User Datagram Protocol
       RFC  793: Transmission Control Protocol
       RFC 2960: Stream Control Transmission Protocol
       RFC 4340: Datagram Congestion Control Protocol (DCCP)
       RFC 4001: Textual Conventions for Internet Network Addresses";
   }

   /*** collection of autonomous system related types ***/

   typedef as-number {
     type uint32;
     description
       "The as-number type represents autonomous system numbers
        which identify an Autonomous System (AS). An AS is a set
        of routers under a single technical administration, using
        an interior gateway protocol and common metrics to route
        packets within the AS, and using an exterior gateway
        protocol to route packets to other ASs'. IANA maintains
        the AS number space and has delegated large parts to the
        regional registries.

        Autonomous system numbers were originally limited to 16
        bits. BGP extensions have enlarged the autonomous system
        number space to 32 bits. This type therefore uses an uint32
        base type without a range restriction in order to support
        a larger autonomous system number space.

        This type is in the value set and its semantics equivalent
        to the InetAutonomousSystemNumber textual convention of
        the SMIv2.";
     reference
      "RFC 1930: Guidelines for creation, selection, and registration
                 of an Autonomous System (AS)
       RFC 4271: A Border Gateway Protocol 4 (BGP-4)
```

```
       RFC 4893: BGP Support for Four-octet AS Number Space
       RFC 4001: Textual Conventions for Internet Network Addresses";
   }

   /*** collection of IP address and hostname related types ***/

   typedef ip-address {
     type union {
       type inet:ipv4-address;
       type inet:ipv6-address;
     }
     description
      "The ip-address type represents an IP address and is IP
       version neutral. The format of the textual representations
       implies the IP version.";
   }

   typedef ipv4-address {
     type string {
       pattern '((0'
             +   '|(1[0-9]{0,2})'
             +   '|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)))'
             +   '|([3-9][0-9]?)'
             +  ')'
             + '\.){3}'
             + '(0'
             +  '|(1[0-9]{0,2})'
             +  '|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)))'
             +  '|([3-9][0-9]?)'
             + ')(%[\p{N}\p{L}]+)?';
     }
     description
       "The ipv4-address type represents an IPv4 address in
        dotted-quad notation. The IPv4 address may include a zone
        index, separated by a % sign.

        The zone index is used to disambiguate identical address
        values.  For link-local addresses, the zone index will
        typically be the interface index number or the name of an
        interface. If the zone index is not present, the default
        zone of the device will be used.

        The canonical format for the zone index is the numerical
        format";
   }

   typedef ipv6-address {
     type string {
```

```
       pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
             + '((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|'
             + '(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}'
             + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
             + '(%[\p{N}\p{L}]+)?';
       pattern '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|'
             + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)'
             + '(%.+)?';
     }
     description
      "The ipv6-address type represents an IPv6 address in full,
       mixed, shortened and shortened mixed notation.  The IPv6
       address may include a zone index, separated by a % sign.

       The zone index is used to disambiguate identical address
       values.  For link-local addresses, the zone index will
       typically be the interface index number or the name of an
       interface. If the zone index is not present, the default
       zone of the device will be used.

       The canonical format of IPv6 addresses uses the compressed
       format described in RFC 4291 section 2.2 item 2 with the
       following additional rules: The :: substitution must be
       applied to the longest sequence of all-zero 16-bit chunks
       in an IPv6 address. If there is a tie, the first sequence
       of all-zero 16-bit chunks is replaced by ::. Single
       all-zero 16-bit chunks are not compressed. The normalized
       format uses lower-case characters and leading zeros are
       not allowed. The canonical format for the zone index is
       the numerical format as described in RFC 4007 section
       11.2.";
     reference
      "RFC 4291: IP Version 6 Addressing Architecture
       RFC 4007: IPv6 Scoped Address Architecture";
   }

   typedef ip-prefix {
     type union {
       type inet:ipv4-prefix;
       type inet:ipv6-prefix;
     }
     description
      "The ip-prefix type represents an IP prefix and is IP
       version neutral. The format of the textual representations
       implies the IP version.";
   }

   typedef ipv4-prefix {
```

```
     type string {
       pattern '(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}'
             + '([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])'
             + '/(([0-9])|([1-2][0-9])|(3[0-2]))';
     }
     description
      "The ipv4-prefix type represents an IPv4 address prefix.
       The prefix length is given by the number following the
       slash character and must be less than or equal to 32.

       A prefix length value of n corresponds to an IP address
       mask which has n contiguous 1-bits from the most
       significant bit (MSB) and all other bits set to 0.

       The canonical format of an IPv4 prefix has all bits of
       the IPv4 address set to zero that are not part of the
       IPv4 prefix.";
   }

   typedef ipv6-prefix {
     type string {
       pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
             + '((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|'
             + '(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}'
             + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
             + '(/(([0-9])|([0-9]{2})|(1[0-1][0-9])|(12[0-8])))';
       pattern '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|'
             + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)'
             + '(/.+)';
     }
     description
      "The ipv6-prefix type represents an IPv6 address prefix.
       The prefix length is given by the number following the
       slash character and must be less than or equal 128.

       A prefix length value of n corresponds to an IP address
       mask which has n contiguous 1-bits from the most
       significant bit (MSB) and all other bits set to 0.

       The IPv6 address should have all bits that do not belong
       to the prefix set to zero.

       The canonical format of an IPv6 prefix has all bits of
       the IPv6 address set to zero that are not part of the
       IPv6 prefix. Furthermore, IPv6 address is represented
       in the compressed format described in RFC 4291 section
       2.2 item 2 with the following additional rules: The ::
       substitution must be applied to the longest sequence of
```

```
        all-zero 16-bit chunks in an IPv6 address. If there is
        a tie, the first sequence of all-zero 16-bit chunks is
        replaced by ::. Single all-zero 16-bit chunks are not
        compressed. The normalized format uses lower-case
        characters and leading zeros are not allowed.";
      reference
       "RFC 4291: IP Version 6 Addressing Architecture";
    }

    /*** collection of domain name and URI types ***/

    typedef domain-name {
      type string {
        pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.)*'
             +  '([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?)'
             +  '|\.';
        length "1..253";
      }
      description
       "The domain-name type represents a DNS domain name. The
        name SHOULD be fully qualified whenever possible.

        Internet domain names are only loosely specified. Section
        3.5 of RFC 1034 recommends a syntax (modified in section
        2.1 of RFC 1123). The pattern above is intended to allow
        for current practise in domain name use, and some possible
        future expansion. It is designed to hold various types of
        domain names, including names used for A or AAAA records
        (host names) and other records, such as SRV records. Note
        that Internet host names have a stricter syntax (described
        in RFC 952) than the DNS recommendations in RFCs 1034 and
        1123, and that systems that want to store host names in
        objects using the domain-name type are recommended to adhere
        to this stricter standard to ensure interoperability.

        The encoding of DNS names in the DNS protocol is limited
        to 255 characters. Since the encoding consists of labels
        prefixed by a length bytes and there is a trailing NULL
        byte, only 253 characters can appear in the textual dotted
        notation.

        The description clause of objects using the domain-name
        type MUST describe how (and when) these names are
        resolved to IP addresses. Note that the resolution of a
        domain-name value may require to query multiple DNS records
        (e.g., A for IPv4 and AAAA for IPv6). The order of the
        resolution process and which DNS record takes precedence
        depends on the configuration of the resolver.
```

```
        The canonical format for domain-name values uses the
        US-ASCII encoding and case-insensitive characters are set
        to lowercase.";
      reference
       "RFC  952: DoD Internet Host Table Specification
        RFC 1034: Domain Names - Concepts and Facilities
        RFC 1123: Requirements for Internet Hosts -- Application
                  and Support
        RFC 3490: Internationalizing Domain Names in Applications
                  (IDNA)";
    }

    typedef host {
      type union {
        type inet:ip-address;
        type inet:domain-name;
      }
      description
       "The host type represents either an IP address or a DNS
        domain name.";
    }

    typedef uri {
      type string;
      description
       "The uri type represents a Uniform Resource Identifier
        (URI) as defined by STD 66.

        Objects using the uri type must be in US-ASCII encoding,
        and MUST be normalized as described by RFC 3986 Sections
        6.2.1, 6.2.2.1, and 6.2.2.2.  All unnecessary
        percent-encoding is removed, and all case-insensitive
        characters are set to lowercase except for hexadecimal
        digits, which are normalized to uppercase as described in
        Section 6.2.2.1.

        The purpose of this normalization is to help provide
        unique URIs.  Note that this normalization is not
        sufficient to provide uniqueness.  Two URIs that are
        textually distinct after this normalization may still be
        equivalent.

        Objects using the uri type may restrict the schemes that
        they permit.  For example, 'data:' and 'urn:' schemes
        might not be appropriate.

        A zero-length URI is not a valid URI.  This can be used to
        express 'URI absent' where required
```

```
         This type is in the value set and its semantics equivalent
         to the Uri SMIv2 textual convention defined in RFC 5017.";
       reference
        "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
         RFC 3305: Report from the Joint W3C/IETF URI Planning Interest
                   Group: Uniform Resource Identifiers (URIs), URLs,
                   and Uniform Resource Names (URNs): Clarifications
                   and Recommendations
         RFC 5017: MIB Textual Conventions for Uniform Resource
                   Identifiers (URIs)";
     }

   }
```

5. IANA Considerations

A registry for standard YANG modules shall be set up. The name of the registry is "IETF YANG Modules" and the registry shall record for each entry the unique name of a YANG module, the assigned XML namespace from the YANG URI Scheme, and a reference to the module's documentation (typically and RFC). Allocations require IETF Review as defined in [RFC5226]. The initial assignments are:

YANG Module	XML namespace	Reference
ietf-yang-types	urn:ietf:params:xml:ns:yang:yang-types	RFC XXXX
ietf-inet-types	urn:ietf:params:xml:ns:yang:inet-types	RFC XXXX

RFC Ed.: replace XXXX with actual RFC number and remove this note

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested.

URI: urn:ietf:params:xml:ns:yang:yang-types
URI: urn:ietf:params:xml:ns:yang:inet-types

Registrant Contact: The NETMOD WG of the IETF.

XML: N/A, the requested URI is an XML namespace.

6. Security Considerations

This document defines common data types using the YANG data modeling language. The definitions themselves have no security impact on the Internet but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [YANG] apply for this document as well.

7. Contributors

The following people all contributed significantly to the initial version of this draft:

- Andy Bierman (andybierman.com)
- Martin Bjorklund (Tail-f Systems)
- Balazs Lengyel (Ericsson)
- David Partain (Ericsson)
- Phil Shafer (Juniper Networks)

8. Acknowledgments

The editor wishes to thank the following individuals for providing helpful comments on various versions of this document: Ladislav Lhotka, Lars-Johan Liman.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[YANG] Bjorklund, M., Ed., "YANG - A data modeling language for NETCONF", draft-ietf-netmod-yang-05 (work in progress).

9.2. Informative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, March 1996.

[RFC2021] Waldbusser, S., "Remote Network Monitoring Management Information Base Version 2 using SMIv2", RFC 2021, January 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.

[RFC2856] Bierman, A., McCloghrie, K., and R. Presuhn, "Textual Conventions for Additional High Capacity Data Types", RFC 2856, June 2000.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3289] Baker, F., Chan, K., and A. Smith, "Management Information Base for the Differentiated Services Architecture", RFC 3289, May 2002.

[RFC3305] Mealling, M. and R. Denenberg, "Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations", RFC 3305, August 2002.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3595] Wijnen, B., "Textual Conventions for IPv6 Flow Label", RFC 3595, September 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, February 2005.

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, March 2005.

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC4893] Vohra, Q. and E. Chen, "BGP Support for Four-octet AS Number Space", RFC 4893, May 2007.

[RFC5017] McWalter, D., "MIB Textual Conventions for Uniform Resource Identifiers (URIs)", RFC 5017, September 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

Appendix A. XSD Translations

This appendix provides XML Schema (XSD) translations of the types defined in this document. This appendix is informative and not normative.

A.1. XSD of Core YANG Derived Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:yang:yang-types"
           xmlns="urn:ietf:params:xml:ns:yang:yang-types"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           version="2009-05-13"
           xml:lang="en"
           xmlns:yang="urn:ietf:params:xml:ns:yang:yang-types">

  <xs:annotation>
    <xs:documentation>
      This module contains a collection of generally useful derived
      YANG data types.

      Copyright (c) 2009 IETF Trust and the persons identified as
      the document authors.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, are permitted provided that the
      following conditions are met:

      - Redistributions of source code must retain the above
        copyright notice, this list of conditions and the
        following disclaimer.

      - Redistributions in binary form must reproduce the above
        copyright notice, this list of conditions and the
        following disclaimer in the documentation and/or other
        materials provided with the distribution.

      - Neither the name of Internet Society, IETF or IETF
        Trust, nor the names of specific contributors, may be
        used to endorse or promote products derived from this
        software without specific prior written permission.

      THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
      CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED
      WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
      WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
```

```
      PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
      OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
      INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
      (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
      GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
      BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
      LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
      (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
      OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
      POSSIBILITY OF SUCH DAMAGE.

      This version of this YANG module is part of RFC XXXX; see
      the RFC itself for full legal notices.
    </xs:documentation>
  </xs:annotation>

  <!-- YANG typedefs -->

  <xs:simpleType name="counter32">
    <xs:annotation>
      <xs:documentation>
        The counter32 type represents a non-negative integer
        which monotonically increases until it reaches a
        maximum value of 2^32-1 (4294967295 decimal), when it
        wraps around and starts increasing again from zero.

        Counters have no defined `initial' value, and thus, a
        single value of a counter has (in general) no information
        content.  Discontinuities in the monotonically increasing
        value normally occur at re-initialization of the
        management system, and at other times as specified in the
        description of an object instance using this type.  If
        such other times can occur, for example, the creation of
        an object instance of type counter32 at times other than
        re-initialization, then a corresponding object should be
        defined, with an appropriate type, to indicate the last
        discontinuity.

        The counter32 type should not be used for configuration
        objects. A default statement should not be used for
        attributes with a type value of counter32.

        This type is in the value set and its semantics equivalent
        to the Counter32 type of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedInt">
```

```
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="zero-based-counter32">
    <xs:annotation>
      <xs:documentation>
        The zero-based-counter32 type represents a counter32
        which has the defined `initial' value zero.

        Objects of this type will be set to zero(0) on creation
        and will thereafter count appropriate events, wrapping
        back to zero(0) when the value 2^32 is reached.

        Provided that an application discovers the new object within
        the minimum time to wrap it can use the initial value as a
        delta since it last polled the table of which this object is
        part.  It is important for a management station to be aware
        of this minimum time and the actual time between polls, and
        to discard data if the actual time is too long or there is
        no defined minimum time.

        This type is in the value set and its semantics equivalent
        to the ZeroBasedCounter32 textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="yang:counter32">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="counter64">
    <xs:annotation>
      <xs:documentation>
        The counter64 type represents a non-negative integer
        which monotonically increases until it reaches a
        maximum value of 2^64-1 (18446744073709551615), when
        it wraps around and starts increasing again from zero.

        Counters have no defined `initial' value, and thus, a
        single value of a counter has (in general) no information
        content.  Discontinuities in the monotonically increasing
        value normally occur at re-initialization of the
        management system, and at other times as specified in the
        description of an object instance using this type.  If
        such other times can occur, for example, the creation of
        an object instance of type counter64 at times other than
        re-initialization, then a corresponding object should be
        defined, with an appropriate type, to indicate the last
```

```
        discontinuity.

        The counter64 type should not be used for configuration
        objects. A default statement should not be used for
        attributes with a type value of counter64.

        This type is in the value set and its semantics equivalent
        to the Counter64 type of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedLong">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="zero-based-counter64">
    <xs:annotation>
      <xs:documentation>
        The zero-based-counter64 type represents a counter64 which
        has the defined `initial' value zero.

        Objects of this type will be set to zero(0) on creation
        and will thereafter count appropriate events, wrapping
        back to zero(0) when the value 2^64 is reached.

        Provided that an application discovers the new object within
        the minimum time to wrap it can use the initial value as a
        delta since it last polled the table of which this object is
        part.  It is important for a management station to be aware
        of this minimum time and the actual time between polls, and
        to discard data if the actual time is too long or there is
        no defined minimum time.

        This type is in the value set and its semantics equivalent
        to the ZeroBasedCounter64 textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="yang:counter64">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="gauge32">
    <xs:annotation>
      <xs:documentation>
        The gauge32 type represents a non-negative integer, which
        may increase or decrease, but shall never exceed a maximum
        value, nor fall below a minimum value.  The maximum value
```

```
        can not be greater than 2^32-1 (4294967295 decimal), and
        the minimum value can not be smaller than 0.  The value of
        a gauge32 has its maximum value whenever the information
        being modeled is greater than or equal to its maximum
        value, and has its minimum value whenever the information
        being modeled is smaller than or equal to its minimum value.
        If the information being modeled subsequently decreases
        below (increases above) the maximum (minimum) value, the
        gauge32 also decreases (increases).

        This type is in the value set and its semantics equivalent
        to the Counter32 type of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedInt">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="gauge64">
    <xs:annotation>
      <xs:documentation>
        The gauge64 type represents a non-negative integer, which
        may increase or decrease, but shall never exceed a maximum
        value, nor fall below a minimum value.  The maximum value
        can not be greater than 2^64-1 (18446744073709551615), and
        the minimum value can not be smaller than 0.  The value of
        a gauge64 has its maximum value whenever the information
        being modeled is greater than or equal to its maximum
        value, and has its minimum value whenever the information
        being modeled is smaller than or equal to its minimum value.
        If the information being modeled subsequently decreases
        below (increases above) the maximum (minimum) value, the
        gauge64 also decreases (increases).

        This type is in the value set and its semantics equivalent
        to the CounterBasedGauge64 SMIv2 textual convention defined
        in RFC 2856
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedLong">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="object-identifier">
    <xs:annotation>
      <xs:documentation>
```

```
        The object-identifier type represents administratively
        assigned names in a registration-hierarchical-name tree.

        Values of this type are denoted as a sequence of numerical
        non-negative sub-identifier values. Each sub-identifier
        value MUST NOT exceed 2^32-1 (4294967295). Sub-identifiers
        are separated by single dots and without any intermediate
        white space.

        Although the number of sub-identifiers is not limited,
        module designers should realize that there may be
        implementations that stick with the SMIv2 limit of 128
        sub-identifiers.

        This type is a superset of the SMIv2 OBJECT IDENTIFIER type
        since it is not restricted to 128 sub-identifiers.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="(([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*)))
                       )(\.(0|([1-9]\d*)))*"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="object-identifier-128">
    <xs:annotation>
      <xs:documentation>
        This type represents object-identifiers restricted to 128
        sub-identifiers.

        This type is in the value set and its semantics equivalent
        to the OBJECT IDENTIFIER type of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="object-identifier">
      <xs:pattern value="\d*(.\d*){1,127}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="date-and-time">
    <xs:annotation>
      <xs:documentation>
        The date-and-time type is a profile of the ISO 8601
        standard for representation of dates and times using the
        Gregorian calendar. The format is most easily described
        using the following ABFN (see RFC 3339):
```

```
        date-fullyear   = 4DIGIT
        date-month      = 2DIGIT  ; 01-12
        date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31
        time-hour       = 2DIGIT  ; 00-23
        time-minute     = 2DIGIT  ; 00-59
        time-second     = 2DIGIT  ; 00-58, 00-59, 00-60
        time-secfrac    = "." 1*DIGIT
        time-numoffset  = ("+" / "-") time-hour ":" time-minute
        time-offset     = "Z" / time-numoffset

        partial-time    = time-hour ":" time-minute ":" time-second
                          [time-secfrac]
        full-date       = date-fullyear "-" date-month "-" date-mday
        full-time       = partial-time time-offset

        date-time       = full-date "T" full-time

        The date-and-time type is consistent with the semantics defined
        in RFC 3339. The data-and-time type is compatible with the
        dateTime XML schema type with the following two notable
        exceptions:

        (a) The data-and-time type does not allow negative years.

        (b) The data-and-time time-offset -00:00 indicates an unknown
            time zone (see RFC 3339) while -00:00 and +00:00 and Z all
            represent the same time zone in dateTime.

        This type is not equivalent to the DateAndTime textual
        convention of the SMIv2 since RFC 3339 uses a different
        separator between full-date and full-time and provides
        higher resolution of time-secfrac.

        The canonical format for date-and-time values mandates the UTC
        time format with the time-offset is indicated by the letter "Z".
        This is consistent with the canonical format used by the
        dateTime XML schema type.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?
                         (Z|(\+|-)\d{2}:\d{2})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="timeticks">
    <xs:annotation>
```

```
      <xs:documentation>
        The timeticks type represents a non-negative integer which
        represents the time, modulo 2^32 (4294967296 decimal), in
        hundredths of a second between two epochs. When objects
        are defined which use this type, the description of the
        object identifies both of the reference epochs.

        This type is in the value set and its semantics equivalent
        to the TimeTicks type of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedInt">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="timestamp">
    <xs:annotation>
      <xs:documentation>
        The timestamp type represents the value of an associated
        timeticks object at which a specific occurrence happened.
        The specific occurrence must be defined in the description
        of any object defined using this type.  When the specific
        occurrence occurred prior to the last time the associated
        timeticks attribute was zero, then the timestamp value is
        zero.  Note that this requires all timestamp values to be
        reset to zero when the value of the associated timeticks
        attribute reaches 497+ days and wraps around to zero.

        The associated timeticks object must be specified
        in the description of any object using this type.

        This type is in the value set and its semantics equivalent
        to the TimeStamp textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="yang:timeticks">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="phys-address">
    <xs:annotation>
      <xs:documentation>
        Represents media- or physical-level addresses represented
        as a sequence octets, each octet represented by two hexadecimal
        numbers. Octets are separated by colons.
```

```
        This type is in the value set and its semantics equivalent
        to the PhysAddress textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a0-fA-F]{2}(:[0-9a0-fA-F]{2})*)?"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="mac-address">
    <xs:annotation>
      <xs:documentation>
        The mac-address type represents an 802 MAC address represented
        in the `canonical' order defined by IEEE 802.1a, i.e., as if it
        were transmitted least significant bit first, even though 802.5
        (in contrast to other 802.x protocols) requires MAC addresses
        to be transmitted most significant bit first.

        This type is in the value set and its semantics equivalent to
        the MacAddress textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="xpath1.0">
    <xs:annotation>
      <xs:documentation>
        This type represents an XPATH 1.0 expression.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>


</xs:schema>
```

A.2. XSD of Internet Specific Derived Types

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:ietf:params:xml:ns:yang:inet-types"
```

```
            xmlns="urn:ietf:params:xml:ns:yang:inet-types"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="2009-05-13"
            xml:lang="en"
            xmlns:inet="urn:ietf:params:xml:ns:yang:inet-types">

   <xs:annotation>
     <xs:documentation>
       This module contains a collection of generally useful derived
       YANG data types for Internet addresses and related things.

       Copyright (c) 2009 IETF Trust and the persons identified as
       the document authors.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, are permitted provided that the
       following conditions are met:

       - Redistributions of source code must retain the above
         copyright notice, this list of conditions and the
         following disclaimer.

       - Redistributions in binary form must reproduce the above
         copyright notice, this list of conditions and the
         following disclaimer in the documentation and/or other
         materials provided with the distribution.

       - Neither the name of Internet Society, IETF or IETF
         Trust, nor the names of specific contributors, may be
         used to endorse or promote products derived from this
         software without specific prior written permission.

       THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED
       WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
       WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
       PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
       OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
       INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
       (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
       GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
       BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
       LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
       (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
       OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
       POSSIBILITY OF SUCH DAMAGE.
```

```
      This version of this YANG module is part of RFC XXXX; see
      the RFC itself for full legal notices.
    </xs:documentation>
  </xs:annotation>

  <!-- YANG typedefs -->

  <xs:simpleType name="ip-version">
    <xs:annotation>
      <xs:documentation>
        This value represents the version of the IP protocol.

        This type is in the value set and its semantics equivalent
        to the InetVersion textual convention of the SMIv2. However,
        the lexical appearance is different from the InetVersion
        textual convention.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:enumeration value="unknown"/>
      <xs:enumeration value="ipv4"/>
      <xs:enumeration value="ipv6"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="dscp">
    <xs:annotation>
      <xs:documentation>
        The dscp type represents a Differentiated Services Code-Point
        that may be used for marking packets in a traffic stream.

        This type is in the value set and its semantics equivalent
        to the Dscp textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedByte">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="63"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ipv6-flow-label">
    <xs:annotation>
      <xs:documentation>
        The flow-label type represents flow identifier or Flow Label
        in an IPv6 packet header that may be used to discriminate
```

```
         traffic flows.

         This type is in the value set and its semantics equivalent
         to the IPv6FlowLabel textual convention of the SMIv2.
       </xs:documentation>
     </xs:annotation>

     <xs:restriction base="xs:unsignedInt">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="1048575"/>
     </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="port-number">
     <xs:annotation>
       <xs:documentation>
         The port-number type represents a 16-bit port number of an
         Internet transport layer protocol such as UDP, TCP, DCCP or
         SCTP. Port numbers are assigned by IANA.  A current list of
         all assignments is available from &lt;http://www.iana.org/&gt;.

         Note that the value zero is not a valid port number. A union
         type might be used in situations where the value zero is
         meaningful.

         This type is in the value set and its semantics equivalent
         to the InetPortNumber textual convention of the SMIv2.
       </xs:documentation>
     </xs:annotation>

     <xs:restriction base="xs:unsignedShort">
       <xs:minInclusive value="1"/>
       <xs:maxInclusive value="65535"/>
     </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="as-number">
     <xs:annotation>
       <xs:documentation>
         The as-number type represents autonomous system numbers
         which identify an Autonomous System (AS). An AS is a set
         of routers under a single technical administration, using
         an interior gateway protocol and common metrics to route
         packets within the AS, and using an exterior gateway
         protocol to route packets to other ASs'. IANA maintains
         the AS number space and has delegated large parts to the
         regional registries.
```

```
        Autonomous system numbers were originally limited to 16
        bits. BGP extensions have enlarged the autonomous system
        number space to 32 bits. This type therefore uses an uint32
        base type without a range restriction in order to support
        a larger autonomous system number space.

        This type is in the value set and its semantics equivalent
        to the InetAutonomousSystemNumber textual convention of
        the SMIv2.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:unsignedInt">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ip-address">
    <xs:annotation>
      <xs:documentation>
        The ip-address type represents an IP address and is IP
        version neutral. The format of the textual representations
        implies the IP version.
      </xs:documentation>
    </xs:annotation>

    <xs:union>
      <xs:simpleType>
        <xs:restriction base="inet:ipv4-address">
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="inet:ipv6-address">
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>

  <xs:simpleType name="ipv4-address">
    <xs:annotation>
      <xs:documentation>
        The ipv4-address type represents an IPv4 address in
        dotted-quad notation. The IPv4 address may include a zone
        index, separated by a % sign.

        The zone index is used to disambiguate identical address
        values.  For link-local addresses, the zone index will
        typically be the interface index number or the name of an
        interface. If the zone index is not present, the default
```

```
          zone of the device will be used.

          The canonical format for the zone index is the numerical
          format
        </xs:documentation>
      </xs:annotation>

      <xs:restriction base="xs:string">
        <xs:pattern value="((0|(1[0-9]{0,2})|(2(([0-4][0-9]?)|(5[0-5]?
                           )|([6-9]?)))|([3-9][0-9]?))\.){3}(0|(1[0-9]{
                           0,2})|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)))|
                           ([3-9][0-9]?))(%[\p{N}\p{L}]+)?"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ipv6-address">
      <xs:annotation>
        <xs:documentation>
          The ipv6-address type represents an IPv6 address in full,
          mixed, shortened and shortened mixed notation.  The IPv6
          address may include a zone index, separated by a % sign.

          The zone index is used to disambiguate identical address
          values.  For link-local addresses, the zone index will
          typically be the interface index number or the name of an
          interface. If the zone index is not present, the default
          zone of the device will be used.

          The canonical format of IPv6 addresses uses the compressed
          format described in RFC 4291 section 2.2 item 2 with the
          following additional rules: The :: substitution must be
          applied to the longest sequence of all-zero 16-bit chunks
          in an IPv6 address. If there is a tie, the first sequence
          of all-zero 16-bit chunks is replaced by ::. Single
          all-zero 16-bit chunks are not compressed. The normalized
          format uses lower-case characters and leading zeros are
          not allowed. The canonical format for the zone index is
          the numerical format as described in RFC 4007 section
          11.2.
        </xs:documentation>
      </xs:annotation>

      <xs:restriction>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|(
                               (([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)(%.
                               +)?"/>
```

```
        </xs:restriction>
      </xs:simpleType>
      <xs:pattern value="((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){
                         0,5}((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4
                         }))|(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])
                         \.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])
                         ))(%[\p{N}\p{L}]+)?"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ip-prefix">
    <xs:annotation>
      <xs:documentation>
        The ip-prefix type represents an IP prefix and is IP
        version neutral. The format of the textual representations
        implies the IP version.
      </xs:documentation>
    </xs:annotation>

    <xs:union>
      <xs:simpleType>
        <xs:restriction base="inet:ipv4-prefix">
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="inet:ipv6-prefix">
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>

  <xs:simpleType name="ipv4-prefix">
    <xs:annotation>
      <xs:documentation>
        The ipv4-prefix type represents an IPv4 address prefix.
        The prefix length is given by the number following the
        slash character and must be less than or equal to 32.

        A prefix length value of n corresponds to an IP address
        mask which has n contiguous 1-bits from the most
        significant bit (MSB) and all other bits set to 0.

        The canonical format of an IPv4 prefix has all bits of
        the IPv4 address set to zero that are not part of the
        IPv4 prefix.
      </xs:documentation>
    </xs:annotation>
```

```
     <xs:restriction base="xs:string">
       <xs:pattern value="(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)
                          {3}([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])/(
                          ([0-9])|([1-2][0-9])|(3[0-2]))"/>
     </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="ipv6-prefix">
     <xs:annotation>
       <xs:documentation>
         The ipv6-prefix type represents an IPv6 address prefix.
         The prefix length is given by the number following the
         slash character and must be less than or equal 128.

         A prefix length value of n corresponds to an IP address
         mask which has n contiguous 1-bits from the most
         significant bit (MSB) and all other bits set to 0.

         The IPv6 address should have all bits that do not belong
         to the prefix set to zero.

         The canonical format of an IPv6 prefix has all bits of
         the IPv6 address set to zero that are not part of the
         IPv6 prefix. Furthermore, IPv6 address is represented
         in the compressed format described in RFC 4291 section
         2.2 item 2 with the following additional rules: The ::
         substitution must be applied to the longest sequence of
         all-zero 16-bit chunks in an IPv6 address. If there is
         a tie, the first sequence of all-zero 16-bit chunks is
         replaced by ::. Single all-zero 16-bit chunks are not
         compressed. The normalized format uses lower-case
         characters and leading zeros are not allowed.
       </xs:documentation>
     </xs:annotation>

     <xs:restriction>
       <xs:simpleType>
         <xs:restriction base="xs:string">
           <xs:pattern value="(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|(
                              (([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)(/.
                              +)"/>
         </xs:restriction>
       </xs:simpleType>
       <xs:pattern value="((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){
                          0,5}((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4
                          }))|(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])
                          \.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])
                          ))(/(([0-9])|([0-9]{2})|(1[0-1][0-9])|(12[0-
```

```
                          8])))"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="domain-name">
      <xs:annotation>
        <xs:documentation>
          The domain-name type represents a DNS domain name. The
          name SHOULD be fully qualified whenever possible.

          Internet domain names are only loosely specified. Section
          3.5 of RFC 1034 recommends a syntax (modified in section
          2.1 of RFC 1123). The pattern above is intended to allow
          for current practise in domain name use, and some possible
          future expansion. It is designed to hold various types of
          domain names, including names used for A or AAAA records
          (host names) and other records, such as SRV records. Note
          that Internet host names have a stricter syntax (described
          in RFC 952) than the DNS recommendations in RFCs 1034 and
          1123, and that systems that want to store host names in
          objects using the domain-name type are recommended to adhere
          to this stricter standard to ensure interoperability.

          The encoding of DNS names in the DNS protocol is limited
          to 255 characters. Since the encoding consists of labels
          prefixed by a length bytes and there is a trailing NULL
          byte, only 253 characters can appear in the textual dotted
          notation.

          The description clause of objects using the domain-name
          type MUST describe how (and when) these names are
          resolved to IP addresses. Note that the resolution of a
          domain-name value may require to query multiple DNS records
          (e.g., A for IPv4 and AAAA for IPv6). The order of the
          resolution process and which DNS record takes precedence
          depends on the configuration of the resolver.

          The canonical format for domain-name values uses the
          US-ASCII encoding and case-insensitive characters are set
          to lowercase.
        </xs:documentation>
      </xs:annotation>

      <xs:restriction base="t0">
        <xs:minLength value="1"/>
        <xs:maxLength value="253"/>
      </xs:restriction>
    </xs:simpleType>
```

```
   <xs:simpleType name="host">
     <xs:annotation>
       <xs:documentation>
         The host type represents either an IP address or a DNS
         domain name.
       </xs:documentation>
     </xs:annotation>

     <xs:union>
       <xs:simpleType>
         <xs:restriction base="inet:ip-address">
         </xs:restriction>
       </xs:simpleType>
       <xs:simpleType>
         <xs:restriction base="inet:domain-name">
         </xs:restriction>
       </xs:simpleType>
     </xs:union>
   </xs:simpleType>

   <xs:simpleType name="uri">
     <xs:annotation>
       <xs:documentation>
         The uri type represents a Uniform Resource Identifier
         (URI) as defined by STD 66.

         Objects using the uri type must be in US-ASCII encoding,
         and MUST be normalized as described by RFC 3986 Sections
         6.2.1, 6.2.2.1, and 6.2.2.2.  All unnecessary
         percent-encoding is removed, and all case-insensitive
         characters are set to lowercase except for hexadecimal
         digits, which are normalized to uppercase as described in
         Section 6.2.2.1.

         The purpose of this normalization is to help provide
         unique URIs.  Note that this normalization is not
         sufficient to provide uniqueness.  Two URIs that are
         textually distinct after this normalization may still be
         equivalent.

         Objects using the uri type may restrict the schemes that
         they permit.  For example, 'data:' and 'urn:' schemes
         might not be appropriate.

         A zero-length URI is not a valid URI.  This can be used to
         express 'URI absent' where required

         This type is in the value set and its semantics equivalent
```

```
         to the Uri SMIv2 textual convention defined in RFC 5017.
       </xs:documentation>
     </xs:annotation>

     <xs:restriction base="xs:string">
     </xs:restriction>
   </xs:simpleType>


   <!-- locally generated simpleType helpers -->

   <xs:simpleType name="t0">
     <xs:restriction base="xs:string">
       <xs:pattern value="((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-z
                          A-Z0-9]\.)*([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,6
                          1})?[a-zA-Z0-9]\.?)|\."/>
     </xs:restriction>
   </xs:simpleType>

 </xs:schema>
```

Appendix B. RelaxNG Translations

This appendix provides RelaxNG translations of the types defined in this document. This appendix is informative and not normative.

B.1. RelaxNG of Core YANG Derived Types

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace dc = "http://purl.org/dc/terms"
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"
namespace nm = "urn:ietf:params:xml:ns:netmod:dsdl-attrib:1"
namespace sch = "http://purl.oclc.org/dsdl/schematron"
namespace yang = "urn:ietf:params:xml:ns:yang:yang-types"

dc:creator [
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
]
dc:description [
  "This module contains a collection of generally useful derived\x{a}" ~
  "YANG data types.\x{a}" ~
  "\x{a}" ~
  "Copyright (c) 2009 IETF Trust and the persons identified as\x{a}" ~
  "the document authors.  All rights reserved.\x{a}" ~
  "\x{a}" ~
  "Redistribution and use in source and binary forms, with or\x{a}" ~
  "without modification, are permitted provided that the\x{a}" ~
  "following conditions are met:\x{a}" ~
  "\x{a}" ~
  "- Redistributions of source code must retain the above\x{a}" ~
  "  copyright notice, this list of conditions and the\x{a}" ~
  "  following disclaimer.\x{a}" ~
  "\x{a}" ~
  "- Redistributions in binary form must reproduce the above\x{a}" ~
  "  copyright notice, this list of conditions and the\x{a}" ~
  "  following disclaimer in the documentation and/or other\x{a}" ~
  "  materials provided with the distribution.\x{a}" ~
  "\x{a}" ~
  "- Neither the name of Internet Society, IETF or IETF\x{a}" ~
  "  Trust, nor the names of specific contributors, may be\x{a}" ~
  "  used to endorse or promote products derived from this\x{a}" ~
  "  software without specific prior written permission.\x{a}" ~
  "\x{a}" ~
  "THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND\x{a}" ~
  "CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED\x{a}" ~
  "WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED\x{a}" ~
  "WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR\x{a}" ~
  "PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT\x{a}" ~
  "OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,\x{a}" ~
```

```
  "INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES\x{a}" ~
  "(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE\x{a}" ~
  "GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR\x{a}" ~
  "BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF\x{a}" ~
  "LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT\x{a}" ~
  "(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT\x{a}" ~
  "OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE\x{a}" ~
  "POSSIBILITY OF SUCH DAMAGE.\x{a}" ~
  "\x{a}" ~
  "This version of this YANG module is part of RFC XXXX; see\x{a}" ~
  "the RFC itself for full legal notices."
]
dc:issued [ "2009-05-13" ]
dc:source [ "YANG module 'ietf-yang-types' (automatic translation)" ]
dc:contributor [
  "WG Web:   <http://tools.ietf.org/wg/netmod/>\x{a}" ~
  "WG List:  <mailto:netmod@ietf.org>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Partain\x{a}" ~
  "          <mailto:david.partain@ericsson.com>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Kessens\x{a}" ~
  "          <mailto: david.kessens@nsn.com>\x{a}" ~
  "\x{a}" ~
  "Editor:   Juergen Schoenwaelder\x{a}" ~
  "          <mailto:j.schoenwaelder@jacobs-university.de>"
]

## The counter32 type represents a non-negative integer
## which monotonically increases until it reaches a
## maximum value of 2^32-1 (4294967295 decimal), when it
## wraps around and starts increasing again from zero.
##
## Counters have no defined `initial' value, and thus, a
## single value of a counter has (in general) no information
## content.  Discontinuities in the monotonically increasing
## value normally occur at re-initialization of the
## management system, and at other times as specified in the
## description of an object instance using this type.  If
## such other times can occur, for example, the creation of
## an object instance of type counter32 at times other than
## re-initialization, then a corresponding object should be
## defined, with an appropriate type, to indicate the last
## discontinuity.
##
## The counter32 type should not be used for configuration
## objects. A default statement should not be used for
## attributes with a type value of counter32.
```

```
##
## This type is in the value set and its semantics equivalent
## to the Counter32 type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
counter32 = xsd:unsignedInt

## The zero-based-counter32 type represents a counter32
## which has the defined `initial' value zero.
##
## Objects of this type will be set to zero(0) on creation
## and will thereafter count appropriate events, wrapping
## back to zero(0) when the value 2^32 is reached.
##
## Provided that an application discovers the new object within
## the minimum time to wrap it can use the initial value as a
## delta since it last polled the table of which this object is
## part.  It is important for a management station to be aware
## of this minimum time and the actual time between polls, and
## to discard data if the actual time is too long or there is
## no defined minimum time.
##
## This type is in the value set and its semantics equivalent
## to the ZeroBasedCounter32 textual convention of the SMIv2.

## See: RFC 2021: Remote Network Monitoring Management Information
##           Base Version 2 using SMIv2
zero-based-counter32 = counter32 >> dsrl:default-content [ "0" ]

## The counter64 type represents a non-negative integer
## which monotonically increases until it reaches a
## maximum value of 2^64-1 (18446744073709551615), when
## it wraps around and starts increasing again from zero.
##
## Counters have no defined `initial' value, and thus, a
## single value of a counter has (in general) no information
## content.  Discontinuities in the monotonically increasing
## value normally occur at re-initialization of the
## management system, and at other times as specified in the
## description of an object instance using this type.  If
## such other times can occur, for example, the creation of
## an object instance of type counter64 at times other than
## re-initialization, then a corresponding object should be
## defined, with an appropriate type, to indicate the last
## discontinuity.
##
## The counter64 type should not be used for configuration
## objects. A default statement should not be used for
```

```
## attributes with a type value of counter64.
##
## This type is in the value set and its semantics equivalent
## to the Counter64 type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
counter64 = xsd:unsignedLong

## The zero-based-counter64 type represents a counter64 which
## has the defined `initial' value zero.
##
## Objects of this type will be set to zero(0) on creation
## and will thereafter count appropriate events, wrapping
## back to zero(0) when the value 2^64 is reached.
##
## Provided that an application discovers the new object within
## the minimum time to wrap it can use the initial value as a
## delta since it last polled the table of which this object is
## part.  It is important for a management station to be aware
## of this minimum time and the actual time between polls, and
## to discard data if the actual time is too long or there is
## no defined minimum time.
##
## This type is in the value set and its semantics equivalent
## to the ZeroBasedCounter64 textual convention of the SMIv2.

## See: RFC 2856: Textual Conventions for Additional High Capacity
##           Data Types
zero-based-counter64 = counter64 >> dsrl:default-content [ "0" ]

## The gauge32 type represents a non-negative integer, which
## may increase or decrease, but shall never exceed a maximum
## value, nor fall below a minimum value.  The maximum value
## can not be greater than 2^32-1 (4294967295 decimal), and
## the minimum value can not be smaller than 0.  The value of
## a gauge32 has its maximum value whenever the information
## being modeled is greater than or equal to its maximum
## value, and has its minimum value whenever the information
## being modeled is smaller than or equal to its minimum value.
## If the information being modeled subsequently decreases
## below (increases above) the maximum (minimum) value, the
## gauge32 also decreases (increases).
##
## This type is in the value set and its semantics equivalent
## to the Counter32 type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
gauge32 = xsd:unsignedInt
```

```
## The gauge64 type represents a non-negative integer, which
## may increase or decrease, but shall never exceed a maximum
## value, nor fall below a minimum value.  The maximum value
## can not be greater than 2^64-1 (18446744073709551615), and
## the minimum value can not be smaller than 0.  The value of
## a gauge64 has its maximum value whenever the information
## being modeled is greater than or equal to its maximum
## value, and has its minimum value whenever the information
## being modeled is smaller than or equal to its minimum value.
## If the information being modeled subsequently decreases
## below (increases above) the maximum (minimum) value, the
## gauge64 also decreases (increases).
##
## This type is in the value set and its semantics equivalent
## to the CounterBasedGauge64 SMIv2 textual convention defined
## in RFC 2856

## See: RFC 2856: Textual Conventions for Additional High Capacity
##           Data Types
gauge64 = xsd:unsignedLong

## The object-identifier type represents administratively
## assigned names in a registration-hierarchical-name tree.
##
## Values of this type are denoted as a sequence of numerical
## non-negative sub-identifier values. Each sub-identifier
## value MUST NOT exceed 2^32-1 (4294967295). Sub-identifiers
## are separated by single dots and without any intermediate
## white space.
##
## Although the number of sub-identifiers is not limited,
## module designers should realize that there may be
## implementations that stick with the SMIv2 limit of 128
## sub-identifiers.
##
## This type is a superset of the SMIv2 OBJECT IDENTIFIER type
## since it is not restricted to 128 sub-identifiers.

## See: ISO/IEC 9834-1: Information technology -- Open Systems
## Interconnection -- Procedures for the operation of OSI
## Registration Authorities: General procedures and top
## arcs of the ASN.1 Object Identifier tree
object-identifier =
  xsd:string {
    pattern =
      "(([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*))))(\.(0|([1-9]\d*)))*"
  }
```

```
## This type represents object-identifiers restricted to 128
## sub-identifiers.
##
## This type is in the value set and its semantics equivalent
## to the OBJECT IDENTIFIER type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
object-identifier-128 = object-identifier

## The date-and-time type is a profile of the ISO 8601
## standard for representation of dates and times using the
## Gregorian calendar. The format is most easily described
## using the following ABFN (see RFC 3339):
##
## date-fullyear   = 4DIGIT
## date-month      = 2DIGIT  ; 01-12
## date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31
## time-hour       = 2DIGIT  ; 00-23
## time-minute     = 2DIGIT  ; 00-59
## time-second     = 2DIGIT  ; 00-58, 00-59, 00-60
## time-secfrac    = "." 1*DIGIT
## time-numoffset  = ("+" / "-") time-hour ":" time-minute
## time-offset     = "Z" / time-numoffset
##
## partial-time    = time-hour ":" time-minute ":" time-second
##                   [time-secfrac]
## full-date       = date-fullyear "-" date-month "-" date-mday
## full-time       = partial-time time-offset
##
## date-time       = full-date "T" full-time
##
## The date-and-time type is consistent with the semantics defined
## in RFC 3339. The data-and-time type is compatible with the
## dateTime XML schema type with the following two notable
## exceptions:
##
## (a) The data-and-time type does not allow negative years.
##
## (b) The data-and-time time-offset -00:00 indicates an unknown
##     time zone (see RFC 3339) while -00:00 and +00:00 and Z all
##     represent the same time zone in dateTime.
##
## This type is not equivalent to the DateAndTime textual
## convention of the SMIv2 since RFC 3339 uses a different
## separator between full-date and full-time and provides
## higher resolution of time-secfrac.
##
## The canonical format for date-and-time values mandates the UTC
```

```
## time format with the time-offset is indicated by the letter "Z".
## This is consistent with the canonical format used by the
## dateTime XML schema type.

## See: RFC 3339: Date and Time on the Internet: Timestamps
## RFC 2579: Textual Conventions for SMIv2
## W3C REC-xmlschema-2-20041028: XML Schema Part 2: Datatypes
##           Second Edition
date-and-time =
  xsd:string {
    pattern =
      "\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|(\+|-)\d{2}:\d{2})"
  }

## The timeticks type represents a non-negative integer which
## represents the time, modulo 2^32 (4294967296 decimal), in
## hundredths of a second between two epochs. When objects
## are defined which use this type, the description of the
## object identifies both of the reference epochs.
##
## This type is in the value set and its semantics equivalent
## to the TimeTicks type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
timeticks = xsd:unsignedInt

## The timestamp type represents the value of an associated
## timeticks object at which a specific occurrence happened.
## The specific occurrence must be defined in the description
## of any object defined using this type.  When the specific
## occurrence occurred prior to the last time the associated
## timeticks attribute was zero, then the timestamp value is
## zero.  Note that this requires all timestamp values to be
## reset to zero when the value of the associated timeticks
## attribute reaches 497+ days and wraps around to zero.
##
## The associated timeticks object must be specified
## in the description of any object using this type.
##
## This type is in the value set and its semantics equivalent
## to the TimeStamp textual convention of the SMIv2.

## See: RFC 2579: Textual Conventions for SMIv2
timestamp = timeticks

## Represents media- or physical-level addresses represented
## as a sequence octets, each octet represented by two hexadecimal
## numbers. Octets are separated by colons.
```

```
##
## This type is in the value set and its semantics equivalent
## to the PhysAddress textual convention of the SMIv2.

## See: RFC 2579: Textual Conventions for SMIv2
phys-address =
  xsd:string { pattern = "([0-9a0-fA-F]{2}(:[0-9a0-fA-F]{2})*)?" }

## The mac-address type represents an 802 MAC address represented
## in the `canonical' order defined by IEEE 802.1a, i.e., as if it
## were transmitted least significant bit first, even though 802.5
## (in contrast to other 802.x protocols) requires MAC addresses
## to be transmitted most significant bit first.
##
## This type is in the value set and its semantics equivalent to
## the MacAddress textual convention of the SMIv2.

## See: RFC 2579: Textual Conventions for SMIv2
mac-address =
  xsd:string { pattern = "[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}" }

## This type represents an XPATH 1.0 expression.

## See: W3C REC-xpath-19991116: XML Path Language (XPath) Version 1.0
xpath1.0 = xsd:string
```

B.2. RelaxNG of Internet Specific Derived Types

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace dc = "http://purl.org/dc/terms"
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"
namespace inet = "urn:ietf:params:xml:ns:yang:inet-types"
namespace nm = "urn:ietf:params:xml:ns:netmod:dsdl-attrib:1"
namespace sch = "http://purl.oclc.org/dsdl/schematron"

dc:creator [
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
]
dc:description [
  "This module contains a collection of generally useful derived\x{a}" ~
  "YANG data types for Internet addresses and related things.\x{a}" ~
  "\x{a}" ~
  "Copyright (c) 2009 IETF Trust and the persons identified as\x{a}" ~
  "the document authors.  All rights reserved.\x{a}" ~
  "\x{a}" ~
  "Redistribution and use in source and binary forms, with or\x{a}" ~
  "without modification, are permitted provided that the\x{a}" ~
  "following conditions are met:\x{a}" ~
```

```
  "\x{a}" ~
  "- Redistributions of source code must retain the above\x{a}" ~
  "  copyright notice, this list of conditions and the\x{a}" ~
  "  following disclaimer.\x{a}" ~
  "\x{a}" ~
  "- Redistributions in binary form must reproduce the above\x{a}" ~
  "  copyright notice, this list of conditions and the\x{a}" ~
  "  following disclaimer in the documentation and/or other\x{a}" ~
  "  materials provided with the distribution.\x{a}" ~
  "\x{a}" ~
  "- Neither the name of Internet Society, IETF or IETF\x{a}" ~
  "  Trust, nor the names of specific contributors, may be\x{a}" ~
  "  used to endorse or promote products derived from this\x{a}" ~
  "  software without specific prior written permission.\x{a}" ~
  "\x{a}" ~
  "THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND\x{a}" ~
  "CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED\x{a}" ~
  "WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED\x{a}" ~
  "WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR\x{a}" ~
  "PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT\x{a}" ~
  "OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,\x{a}" ~
  "INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES\x{a}" ~
  "(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE\x{a}" ~
  "GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR\x{a}" ~
  "BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF\x{a}" ~
  "LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT\x{a}" ~
  "(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT\x{a}" ~
  "OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE\x{a}" ~
  "POSSIBILITY OF SUCH DAMAGE.\x{a}" ~
  "\x{a}" ~
  "This version of this YANG module is part of RFC XXXX; see \x{a}" ~
  "the RFC itself for full legal notices."
]
dc:issued [ "2009-05-13" ]
dc:source [ "YANG module 'ietf-inet-types' (automatic translation)" ]
dc:contributor [
  "WG Web:   <http://tools.ietf.org/wg/netmod/>\x{a}" ~
  "WG List:  <mailto:netmod@ietf.org>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Partain\x{a}" ~
  "          <mailto:david.partain@ericsson.com>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Kessens\x{a}" ~
  "          <mailto:david.kessens@nsn.com>\x{a}" ~
  "\x{a}" ~
  "Editor:   Juergen Schoenwaelder\x{a}" ~
  "          <mailto:j.schoenwaelder@jacobs-university.de>"
]
```

```
## This value represents the version of the IP protocol.
##
## This type is in the value set and its semantics equivalent
## to the InetVersion textual convention of the SMIv2. However,
## the lexical appearance is different from the InetVersion
## textual convention.

## See: RFC  791: Internet Protocol
## RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
## RFC 4001: Textual Conventions for Internet Network Addresses
ip-version = "unknown" | "ipv4" | "ipv6"

## The dscp type represents a Differentiated Services Code-Point
## that may be used for marking packets in a traffic stream.
##
## This type is in the value set and its semantics equivalent
## to the Dscp textual convention of the SMIv2.

## See: RFC 3289: Management Information Base for the Differentiated
##           Services Architecture
## RFC 2474: Definition of the Differentiated Services Field
##           (DS Field) in the IPv4 and IPv6 Headers
## RFC 2780: IANA Allocation Guidelines For Values In
##           the Internet Protocol and Related Headers
dscp = xsd:unsignedByte { minInclusive = "0" maxInclusive = "63" }

## The flow-label type represents flow identifier or Flow Label
## in an IPv6 packet header that may be used to discriminate
## traffic flows.
##
## This type is in the value set and its semantics equivalent
## to the IPv6FlowLabel textual convention of the SMIv2.

## See: RFC 3595: Textual Conventions for IPv6 Flow Label
## RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
ipv6-flow-label =
  xsd:unsignedInt { minInclusive = "0" maxInclusive = "1048575" }

## The port-number type represents a 16-bit port number of an
## Internet transport layer protocol such as UDP, TCP, DCCP or
## SCTP. Port numbers are assigned by IANA.  A current list of
## all assignments is available from <http://www.iana.org/>.
##
## Note that the value zero is not a valid port number. A union
## type might be used in situations where the value zero is
## meaningful.
##
## This type is in the value set and its semantics equivalent
```

```
## to the InetPortNumber textual convention of the SMIv2.

## See: RFC  768: User Datagram Protocol
## RFC  793: Transmission Control Protocol
## RFC 2960: Stream Control Transmission Protocol
## RFC 4340: Datagram Congestion Control Protocol (DCCP)
## RFC 4001: Textual Conventions for Internet Network Addresses
port-number =
  xsd:unsignedShort { minInclusive = "1" maxInclusive = "65535" }

## The as-number type represents autonomous system numbers
## which identify an Autonomous System (AS). An AS is a set
## of routers under a single technical administration, using
## an interior gateway protocol and common metrics to route
## packets within the AS, and using an exterior gateway
## protocol to route packets to other ASs'. IANA maintains
## the AS number space and has delegated large parts to the
## regional registries.
##
## Autonomous system numbers were originally limited to 16
## bits. BGP extensions have enlarged the autonomous system
## number space to 32 bits. This type therefore uses an uint32
## base type without a range restriction in order to support
## a larger autonomous system number space.
##
## This type is in the value set and its semantics equivalent
## to the InetAutonomousSystemNumber textual convention of
## the SMIv2.

## See: RFC 1930: Guidelines for creation, selection, and registration
##           of an Autonomous System (AS)
## RFC 4271: A Border Gateway Protocol 4 (BGP-4)
## RFC 4893: BGP Support for Four-octet AS Number Space
## RFC 4001: Textual Conventions for Internet Network Addresses
as-number = xsd:unsignedInt

## The ip-address type represents an IP address and is IP
## version neutral. The format of the textual representations
## implies the IP version.
ip-address = ipv4-address | ipv6-address

## The ipv4-address type represents an IPv4 address in
## dotted-quad notation. The IPv4 address may include a zone
## index, separated by a % sign.
##
## The zone index is used to disambiguate identical address
## values.  For link-local addresses, the zone index will
## typically be the interface index number or the name of an
```

```
## interface. If the zone index is not present, the default
## zone of the device will be used.
##
## The canonical format for the zone index is the numerical
## format
ipv4-address =
  xsd:string {
    pattern =
      "((0|(1[0-9]{0,2})|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)"
    ~ "))|([3-9][0-9]?))\.){3}(0|(1[0-9]{0,2})|(2(([0-4][0-9]?)|(5["
    ~ "0-5]?)|([6-9]?)))|([3-9][0-9]?))(%[\p{N}\p{L}]+)?"
  }

## The ipv6-address type represents an IPv6 address in full,
## mixed, shortened and shortened mixed notation.  The IPv6
## address may include a zone index, separated by a % sign.
##
## The zone index is used to disambiguate identical address
## values.  For link-local addresses, the zone index will
## typically be the interface index number or the name of an
## interface. If the zone index is not present, the default
## zone of the device will be used.
##
## The canonical format of IPv6 addresses uses the compressed
## format described in RFC 4291 section 2.2 item 2 with the
## following additional rules: The :: substitution must be
## applied to the longest sequence of all-zero 16-bit chunks
## in an IPv6 address. If there is a tie, the first sequence
## of all-zero 16-bit chunks is replaced by ::. Single
## all-zero 16-bit chunks are not compressed. The normalized
## format uses lower-case characters and leading zeros are
## not allowed. The canonical format for the zone index is
## the numerical format as described in RFC 4007 section
## 11.2.

## See: RFC 4291: IP Version 6 Addressing Architecture
## RFC 4007: IPv6 Scoped Address Architecture
ipv6-address =
  xsd:string {
    pattern =
      "((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-"
    ~ "9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0-9]"
    ~ "|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9"
    ~ "])))(%[\p{N}\p{L}]+)?"
    pattern =
      "(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|((([^:]+:)*[^:]"
    ~ "+)?::(([^:]+:)*[^:]+)?)(%.+)?"
  }
```

```
## The ip-prefix type represents an IP prefix and is IP
## version neutral. The format of the textual representations
## implies the IP version.
ip-prefix = ipv4-prefix | ipv6-prefix

## The ipv4-prefix type represents an IPv4 address prefix.
## The prefix length is given by the number following the
## slash character and must be less than or equal to 32.
##
## A prefix length value of n corresponds to an IP address
## mask which has n contiguous 1-bits from the most
## significant bit (MSB) and all other bits set to 0.
##
## The canonical format of an IPv4 prefix has all bits of
## the IPv4 address set to zero that are not part of the
## IPv4 prefix.
ipv4-prefix =
  xsd:string {
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-1]?"
    ~ "[0-9]?[0-9]|2[0-4][0-9]|25[0-5])/(([0-9])|([1-2][0-9])|(3[0-"
    ~ "2]))"
  }

## The ipv6-prefix type represents an IPv6 address prefix.
## The prefix length is given by the number following the
## slash character and must be less than or equal 128.
##
## A prefix length value of n corresponds to an IP address
## mask which has n contiguous 1-bits from the most
## significant bit (MSB) and all other bits set to 0.
##
## The IPv6 address should have all bits that do not belong
## to the prefix set to zero.
##
## The canonical format of an IPv6 prefix has all bits of
## the IPv6 address set to zero that are not part of the
## IPv6 prefix. Furthermore, IPv6 address is represented
## in the compressed format described in RFC 4291 section
## 2.2 item 2 with the following additional rules: The ::
## substitution must be applied to the longest sequence of
## all-zero 16-bit chunks in an IPv6 address. If there is
## a tie, the first sequence of all-zero 16-bit chunks is
## replaced by ::. Single all-zero 16-bit chunks are not
## compressed. The normalized format uses lower-case
## characters and leading zeros are not allowed.

## See: RFC 4291: IP Version 6 Addressing Architecture
```

```
ipv6-prefix =
  xsd:string {
    pattern =
      "((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-"
    ~ "9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0-9]"
    ~ "|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9"
    ~ "])))(/(([0-9])|([0-9]{2})|(1[0-1][0-9])|(12[0-8])))"
    pattern =
      "(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|((([^:]+:)*[^:]"
    ~ "+)?::(([^:]+:)*[^:]+)?)(/.+)"
  }

## The domain-name type represents a DNS domain name. The
## name SHOULD be fully qualified whenever possible.
##
## Internet domain names are only loosely specified. Section
## 3.5 of RFC 1034 recommends a syntax (modified in section
## 2.1 of RFC 1123). The pattern above is intended to allow
## for current practise in domain name use, and some possible
## future expansion. It is designed to hold various types of
## domain names, including names used for A or AAAA records
## (host names) and other records, such as SRV records. Note
## that Internet host names have a stricter syntax (described
## in RFC 952) than the DNS recommendations in RFCs 1034 and
## 1123, and that systems that want to store host names in
## objects using the domain-name type are recommended to adhere
## to this stricter standard to ensure interoperability.
##
## The encoding of DNS names in the DNS protocol is limited
## to 255 characters. Since the encoding consists of labels
## prefixed by a length bytes and there is a trailing NULL
## byte, only 253 characters can appear in the textual dotted
## notation.
##
## The description clause of objects using the domain-name
## type MUST describe how (and when) these names are
## resolved to IP addresses. Note that the resolution of a
## domain-name value may require to query multiple DNS records
## (e.g., A for IPv4 and AAAA for IPv6). The order of the
## resolution process and which DNS record takes precedence
## depends on the configuration of the resolver.
##
## The canonical format for domain-name values uses the
## US-ASCII encoding and case-insensitive characters are set
## to lowercase.

## See: RFC  952: DoD Internet Host Table Specification
## RFC 1034: Domain Names - Concepts and Facilities
```

```
## RFC 1123: Requirements for Internet Hosts -- Application
##           and Support
## RFC 3490: Internationalizing Domain Names in Applications
##           (IDNA)
domain-name =
  xsd:string {
    pattern =
      "((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.)"
    ~ "*([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?)|\."
    minLength = "1"
    maxLength = "253"
  }

## The host type represents either an IP address or a DNS
## domain name.
host = ip-address | domain-name

## The uri type represents a Uniform Resource Identifier
## (URI) as defined by STD 66.
##
## Objects using the uri type must be in US-ASCII encoding,
## and MUST be normalized as described by RFC 3986 Sections
## 6.2.1, 6.2.2.1, and 6.2.2.2.  All unnecessary
## percent-encoding is removed, and all case-insensitive
## characters are set to lowercase except for hexadecimal
## digits, which are normalized to uppercase as described in
## Section 6.2.2.1.
##
## The purpose of this normalization is to help provide
## unique URIs.  Note that this normalization is not
## sufficient to provide uniqueness.  Two URIs that are
## textually distinct after this normalization may still be
## equivalent.
##
## Objects using the uri type may restrict the schemes that
## they permit.  For example, 'data:' and 'urn:' schemes
## might not be appropriate.
##
## A zero-length URI is not a valid URI.  This can be used to
## express 'URI absent' where required
##
## This type is in the value set and its semantics equivalent
## to the Uri SMIv2 textual convention defined in RFC 5017.

## See: RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
## RFC 3305: Report from the Joint W3C/IETF URI Planning Interest
##           Group: Uniform Resource Identifiers (URIs), URLs,
##           and Uniform Resource Names (URNs): Clarifications
```

```
##           and Recommendations
## RFC 5017: MIB Textual Conventions for Uniform Resource
##           Identifiers (URIs)
uri = xsd:string
```

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de