Network Working Group J. Schoenwaelder, Ed.
Internet-Draft Jacobs University
Obsoletes: 6021 (if approved) March 25, 2013
Intended status: Standards Track
Expires: September 26, 2013

Common YANG Data Types
draft-ietf-netmod-rfc6021-bis-01

Abstract

This document introduces a collection of common data types to be used with the YANG data modeling language. This document obsoletes RFC 6021.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF) [RFC6241]. 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 derived types are designed to be applicable for modeling all areas of management information. 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.

This version of the document adds new type definitions to the YANG modules and obsoletes [RFC6021]. For the further details, see the revision statement of the YANG modules.

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 of the types defined in subsequent sections and their equivalent Structure of Management Information Version 2 (SMIv2) [RFC2578][RFC2579] data types. A YANG data type is equivalent to an SMIv2 data type if the data types have the same set of values and the semantics of the values are equivalent.

Table 1 lists the types defined in the ietf-yang-types YANG module and the corresponding SMIv2 types (- indicates there is no corresponding SMIv2 type).

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
yang-identifier	-
date-and-time	-
timeticks	TimeTicks (SNMPv2-SMI)
timestamp	TimeStamp (SNMPv2-TC)
phys-address	PhysAddress (SNMPv2-TC)
mac-address	MacAddress (SNMPv2-TC)
xpath1.0	-
hex-string	-
uuid	-
dotted-quad	-

Table 1: ietf-yang-types

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

YANG type	Equivalent SMIv2 type (module)
ip-version	InetVersion (INET-ADDRESS-MIB)
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-address-no-zone	-
ipv4-address-no-zone	-
ipv6-address-no-zone	-
ip-prefix	-
ipv4-prefix	-
ipv6-prefix	-
domain-name	-
host	-
uri	Uri (URI-TC-MIB)

Table 2: ietf-inet-types

3. Core YANG Derived Types

The ietf-yang-types YANG module references [IEEE802], [ISO9834-1], [RFC2578], [RFC2579], [RFC2856], [RFC3339], [RFC4122], [RFC4502], [RFC6020], [XPATH], and [XSD-TYPES].

```
<CODE BEGINS> file "ietf-yang-types@2013-03-25.yang"

module ietf-yang-types {

  namespace "urn:ietf:params:xml:ns:yang:ietf-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 Kessens
              <mailto:david.kessens@nsn.com>

    WG Chair: Juergen Schoenwaelder
              <mailto:j.schoenwaelder@jacobs-university.de>

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

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

    Copyright (c) 2013 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";

  revision 2013-03-25 {
    description
```

```
        "This revision adds the following new data types:
         - yang-identifier
         - hex-string
         - uuid
         - dotted-quad";
       reference
        "RFC XXXX: Common YANG Data Types";
     }

     revision 2010-09-24 {
       description
        "Initial revision.";
       reference
        "RFC 6021: Common YANG Data Types";
     }

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

     typedef counter32 {
       type uint32;
       description
        "The counter32 type represents a non-negative integer
         that 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 a schema node using this type.  If such
         other times can occur, for example, the creation of
         a schema node of type counter32 at times other than
         re-initialization, then a corresponding schema node
         should be defined, with an appropriate type, to indicate
         the last discontinuity.

         The counter32 type should not be used for configuration
         schema nodes.  A default statement SHOULD NOT be used in
         combination with the type counter32.

         In the value set and its semantics, this type is 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
         that has the defined 'initial' value zero.

         A schema node of this type will be set to zero (0) on creation
         and will thereafter increase monotonically until it reaches
         a maximum value of 2^32-1 (4294967295 decimal), when it
         wraps around and starts increasing again from zero.

         Provided that an application discovers a new schema node
         of this type within the minimum time to wrap, it can use the
         'initial' value as a delta.  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.

         In the value set and its semantics, this type is equivalent
         to the ZeroBasedCounter32 textual convention of the SMIv2.";
       reference
         "RFC 4502: Remote Network Monitoring Management Information
                    Base Version 2";
     }

     typedef counter64 {
       type uint64;
       description
        "The counter64 type represents a non-negative integer
         that monotonically increases until it reaches a
         maximum value of 2^64-1 (18446744073709551615 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 a schema node using this type.  If such
         other times can occur, for example, the creation of
         a schema node of type counter64 at times other than
         re-initialization, then a corresponding schema node
         should be defined, with an appropriate type, to indicate
         the last discontinuity.

         The counter64 type should not be used for configuration
         schema nodes.  A default statement SHOULD NOT be used in
```

```
         combination with the type counter64.

         In the value set and its semantics, this type is 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 that
         has the defined 'initial' value zero.

         A schema node of this type will be set to zero (0) on creation
         and will thereafter increase monotonically until it reaches
         a maximum value of 2^64-1 (18446744073709551615 decimal),
         when it wraps around and starts increasing again from zero.

         Provided that an application discovers a new schema node
         of this type within the minimum time to wrap, it can use the
         'initial' value as a delta.  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.

         In the value set and its semantics, this type is 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
         cannot be greater than 2^32-1 (4294967295 decimal), and
         the minimum value cannot 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).

         In the value set and its semantics, this type is equivalent
         to the Gauge32 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
         cannot be greater than 2^64-1 (18446744073709551615), and
         the minimum value cannot 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).

         In the value set and its semantics, this type is 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
         whitespace.

         The ASN.1 standard restricts the value space of the first
         sub-identifier to 0, 1, or 2.  Furthermore, the value space
         of the second sub-identifier is restricted to the range
         0 to 39 if the first sub-identifier is 0 or 1.  Finally,
         the ASN.1 standard requires that an object identifier
         has always at least two sub-identifier.  The pattern
         captures these restrictions.

         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.  Hence,
         this type SHOULD NOT be used to represent the SMIv2 OBJECT
         IDENTIFIER type, the object-identifier-128 type SHOULD be
         used instead.";
       reference
        "ISO9834-1: 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.

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

     typedef yang-identifier {
       type string {
         length "1..max";
         pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
         pattern '.|..|[^xX].*|.[^mM].*|..[^lL].*';
```

```
    }
    description
      "A YANG identifier string as defined in RFC 6020, page 163.
       An identifier must start with an alphabetic character or
       an underscore followed by an arbitrary sequence of
       alphabetic or numeric characters, underscores, hyphens
       or dots.

       A YANG identifier MUST NOT start with any possible
       combination of the lower-case or upper-case character
       sequence 'xml'.";
    reference
      "RFC 6020: YANG - A Data Modeling Language for the Network
                 Configuration Protocol (NETCONF)";
  }

  /*** 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 profile is defined by the
      date-time production in Section 5.6 of RFC 3339.

      The date-and-time type is compatible with the dateTime XML
      schema type with the following notable exceptions:

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

      (b) The date-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.

      (c) The canonical format (see below) of data-and-time values
          differs from the canonical format used by the dateTime XML
          schema type, which requires all times to be in UTC using
          the time-offset 'Z'.

      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 with a known time
         zone uses a numeric time zone offset that is calculated using
         the device's configured known offset to UTC time.  A change of
         the device's offset to UTC time will cause date-and-time values
         to change accordingly.  Such changes might happen periodically
         in case a server follows automatically daylight saving time
         (DST) time zone offset changes.  The canonical format for
         date-and-time values with an unknown time zone (usually
         referring to the notion of local time) uses the time-offset
         -00:00.";
       reference
        "RFC 3339: Date and Time on the Internet: Timestamps
         RFC 2579: Textual Conventions for SMIv2
         XSD-TYPES: XML Schema Part 2: Datatypes Second Edition";
     }

     typedef timeticks {
       type uint32;
       description
        "The timeticks type represents a non-negative integer that
         represents the time, modulo 2^32 (4294967296 decimal), in
         hundredths of a second between two epochs.  When a schema
         node is defined that uses this type, the description of
         the schema node identifies both of the reference epochs.

         In the value set and its semantics, this type is 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 schema node at which a specific occurrence
         happened. The specific occurrence must be defined in the
         description of any schema node 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 schema node must be specified
         in the description of any schema node using this type.
```

```
       In the value set and its semantics, this type is 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 '([0-9a-fA-F]{2}(:[0-9a-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.  The canonical
       representation uses lowercase characters.

       In the value set and its semantics, this type is 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 IEEE 802 MAC address.
       The canonical representation uses lowercase characters.

       In the value set and its semantics, this type is equivalent
       to the MacAddress textual convention of the SMIv2.";
     reference
      "IEEE 802: IEEE Standard for Local and Metropolitan Area
                 Networks: Overview and Architecture
       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.

       When a schema node is defined that uses this type, the
```

```
         description of the schema node MUST specify the XPath
         context in which the XPath expression is evaluated.";
       reference
        "XPATH: XML Path Language (XPath) Version 1.0";
     }

     /*** collection of string types ***/

     typedef hex-string {
       type string {
         pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*';
       }
       description
        "A hexadecimal string with octets represented as hex digits
         separated by colons.  The canonical representation uses
         lowercase characters.";
     }

     typedef uuid {
       type string {
         pattern '[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-'
               + '[0-9a-fA-F]{4}-[0-9a-fA-F]{12}';
       }
       description
        "A Universally Unique IDentifier in the string representation
         defined in RFC 4122.  The canonical representation uses
         lowercase characters.

         The following is an example of a UUID in string representation:
         f81d4fae-7dec-11d0-a765-00a0c91e6bf6
         ";
       reference
        "RFC 4122: A Universally Unique IDentifier (UUID) URN
                   Namespace";
     }

     typedef dotted-quad {
       type string {
         pattern
           '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
         + '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])';
       }
       description
         "An unsigned 32-bit number expressed in the dotted-quad
          notation, i.e., four octets written as decimal numbers
          and separated with the '.' (full stop) character.";
     }
   }
```

<CODE ENDS>

4. Internet-Specific Derived Types

The ietf-inet-types YANG module references [RFC0768], [RFC0791], [RFC0793], [RFC0952], [RFC1034], [RFC1123], [RFC1930], [RFC2460], [RFC2474], [RFC2780], [RFC2782], [RFC3289], [RFC3305], [RFC3492], [RFC3595], [RFC3986], [RFC4001], [RFC4007], [RFC4271], [RFC4291], [RFC4340], [RFC4960], [RFC5017], [RFC5891], [RFC5952], and [RFC6793].

```
<CODE BEGINS> file "ietf-inet-types@2013-03-25.yang"

module ietf-inet-types {

  namespace "urn:ietf:params:xml:ns:yang:ietf-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 Kessens
              <mailto:david.kessens@nsn.com>

    WG Chair: Juergen Schoenwaelder
              <mailto:j.schoenwaelder@jacobs-university.de>

    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) 2013 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";
```

```
     revision 2013-03-25 {
       description
        "This revision adds the following new data types:
         - ip-address-no-zone
         - ipv4-address-no-zone
         - ipv6-address-no-zone";
       reference
        "RFC XXXX: Common YANG Data Types";
     }

     revision 2010-09-24 {
       description
        "Initial revision.";
       reference
        "RFC 6021: Common YANG Data Types";
     }

     /*** 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.

         In the value set and its semantics, this type is equivalent
         to the InetVersion textual convention of the SMIv2.";
       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.

         In the value set and its semantics, this type is 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.

         In the value set and its semantics, this type is 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 "0..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 port number value zero is reserved by IANA.  In
         situations where the value zero does not make sense, it can
         be excluded by subtyping the port-number type.
```

```
         In the value set and its semantics, this type is equivalent
         to the InetPortNumber textual convention of the SMIv2.";
       reference
        "RFC  768: User Datagram Protocol
         RFC  793: Transmission Control Protocol
         RFC 4960: 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.

         In the value set and its semantics, this type is 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 4001: Textual Conventions for Internet Network Addresses
         RFC 6793: BGP Support for Four-octet AS Number Space";
     }

     /*** 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 representation
         implies the IP version.  This type supports scoped addresses
         by allowing zone identifiers in the address format.";
       reference
        "RFC 4007: IPv6 Scoped Address Architecture";
     }

     typedef ipv4-address {
       type string {
         pattern
           '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
         +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'
         + '(%[\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 canonical
         format uses lowercase 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
         RFC 5952: A Recommendation for IPv6 Address Text
                   Representation";
     }

     typedef ip-address-no-zone {
       type union {
         type inet:ipv4-address-no-zone;
         type inet:ipv6-address-no-zone;
       }
       description
        "The ip-address-no-zone type represents an IP address and is
         IP version neutral.  The format of the textual representation
         implies the IP version.  This type does not support scoped
         addresses since it does not allow zone identifiers in the
         address format.";
       reference
        "RFC 4007: IPv6 Scoped Address Architecture";
     }

     typedef ipv4-address-no-zone {
       type inet:ipv4-address {
         pattern '[\.0-9]*';
       }
       description
         "An IPv4 address without a zone index. This type may be used
          in situations where the zone is known from the context and
          hence no zone index is needed.";
     }
```

```
     typedef ipv6-address-no-zone {
       type inet:ipv6-address {
         pattern '[0-9a-fA-F:]*';
       }
       description
         "An IPv6 address without a zone index. This type may be used
          in situations where the zone is known from the context and
          hence no zone index is needed.";
       reference
        "RFC 4291: IP Version 6 Addressing Architecture
         RFC 4007: IPv6 Scoped Address Architecture
         RFC 5952: A Recommendation for IPv6 Address Text
                   Representation";
     }

     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
            '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
          +  '([0-9]|[1-9][0-9]|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 that 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 that 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 canonical format uses lowercase
         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 practice 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
         schema nodes 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 schema nodes using the domain-name
         type MUST describe when and how 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 can either be defined
         explicitely or it may depend on the configuration of the
         resolver.

         Domain-name values use the US-ASCII encoding.  Their canonical
         format uses lowercase US-ASCII characters.  Internationalized
         domain names MUST be encoded in punycode as described in RFC
         3492";
       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 2782: A DNS RR for specifying the location of services
                   (DNS SRV)
         RFC 3492: Punycode: A Bootstring encoding of Unicode for
                   Internationalized Domain Names in Applications
                   (IDNA)
         RFC 5891: Internationalizing Domain Names in Applications
                   (IDNA): Protocol";
     }
```

```
     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.

         In the value set and its semantics, this type is 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)";
     }
```

```
   }

   <CODE ENDS>
```

5. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registrations have been made.

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

Registrant Contact: The NETMOD WG of the IETF.

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

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

Registrant Contact: The NETMOD WG of the IETF.

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

This document registers two YANG modules in the YANG Module Names registry [RFC6020].

```
name:         ietf-yang-types
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-types
prefix:       yang
reference:    RFC 6021

name:         ietf-inet-types
namespace:    urn:ietf:params:xml:ns:yang:ietf-inet-types
prefix:       inet
reference:    RFC 6021
```

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 [RFC6020] apply for this document as well.

7. Contributors

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

- Andy Bierman (Brocade)
- 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: Andy Bierman, Martin Bjorklund, Ladislav Lhotka, Lars-Johan Liman, and Dan Romascanu.

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

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.

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

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

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

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

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

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

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

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

9.2. Informative References

[IEEE802] IEEE, "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture", IEEE Std. 802-2001.

[ISO9834-1]
ISO/IEC, "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", ISO/IEC 9834-1:2008, 2008.

[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.

[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.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for

specifying the location of services (DNS SRV)", RFC 2782, February 2000.

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

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

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

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

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

[RFC4502] Waldbusser, S., "Remote Network Monitoring Management Information Base Version 2", RFC 4502, May 2006.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

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

[RFC5891] Klensin, J., "Internationalizing Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

[RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "NETCONF Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, December 2012.

[XSD-TYPES] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

Appendix A. Changes from RFC 6021

This version adds new type definitions to the YANG modules. The following new data types have been added to the ietf-yang-types module:

- yang-identifier
- hex-string
- uuid
- dotted-quad

The following new data types have been added to the ietf-inet-types module:

- ip-address-no-zone
- ipv4-address-no-zone
- ipv6-address-no-zone

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de