Network Working Group J. Schoenwaelder, Ed.
Internet-Draft Jacobs University
Obsoletes: 6991 (if approved) January 15, 2022
Intended status: Standards Track
Expires: July 19, 2022

Common YANG Data Types
draft-ietf-netmod-rfc6991-bis-10

Abstract

This document introduces a collection of common data types to be used with the YANG data modeling language. This document obsoletes RFC 6991.

Status of This Memo

This Internet-Draft will expire on July 19, 2022.

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC7950] 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 document adds new type definitions to the YANG modules and obsoletes [RFC6991]. For further details, see the revision

statements of the YANG modules in Section 3 and Section 4 and the summary in Appendix A.

This document uses the YANG terminology defined in Section 3 of [RFC7950].

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] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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
date-and-time	-
date	-
time	-
hours32	-
minutes32	-
seconds32	-
centiseconds32	TimeInterval (SNMPv2-TC)
milliseconds32	-
microseconds32	-
microseconds64	-
nanoseconds32	-
nanoseconds64	-
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	-
yang-identifier	-
revision-identifier	-
percent	-
percent-i32	-
percent-u32	-

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-name	-
host	-
uri	Uri (URI-TC-MIB)
email-address	-

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], [RFC7950], [RFC8294], [XPATH], and [XSD-TYPES].

<CODE BEGINS> file "ietf-yang-types@2022-01-15.yang"

```
module ietf-yang-types {

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

  organization
   "IETF Network Modeling (NETMOD) Working Group";

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

    Editor:   Juergen Schoenwaelder
```

```
              <mailto:j.schoenwaelder@jacobs-university.de>";

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

    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 (RFC 2119) (RFC 8174) when, and only when,
    they appear in all capitals, as shown here.

    Copyright (c) 2022 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
    (https://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 2022-01-15 {
    description
     "This revision adds the following new data types:
      - date, time
      - hours32, minutes32, seconds32, centiseconds32, milliseconds32,
      - microseconds32, microseconds64, nanoseconds32, nanoseconds64
      - revision-identifier
      - percent, percent-i32, percent-u32
      The yang-identifier definition has been aligned with YANG 1.1.";
    reference
     "RFC XXXX: Common YANG Data Types";
  }

  revision 2013-07-15 {
    description
     "This revision adds the following new data types:
      - yang-identifier
      - hex-string
      - uuid
      - dotted-quad";
    reference
     "RFC 6991: 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 instantiation 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 instance 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
      instance 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 instantiation 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 instance 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
      instance 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][0-9]*))))'
            + '(\.(0|([1-9][0-9]*)))*';
    }
    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-identifiers.  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 '[0-9]*(\.[0-9]*){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)";
   }

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

   typedef date-and-time {
     type string {
       pattern '[0-9]{4}-(1[0-2]|0[1-9])-(0[1-9]|[1|2][0-9]|3[0-1])'
             + 'T(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](\.[0-9]+)?'
             + '(Z|[\+\-]((1[0-3]|0[0-9]):([0-5][0-9])|14:00))?';
     }
     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 dateTime type with the following notable exceptions:

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

      (b) The time-offset -00:00 indicates that the date-and-time
          value is reported in UTC and that the local time zone
          reference point is unknown. The time-offsets +00:00 and Z
          both indicate that the date-and-time value is reported in
          UTC and that the local time reference point is UTC (see RFC
          3339 section 4.3).

      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, i.e., date-and-time values must be reported in UTC.";
    reference
     "RFC 3339: Date and Time on the Internet: Timestamps
      RFC 2579: Textual Conventions for SMIv2
      XSD-TYPES: XML Schema Definition Language (XSD) 1.1
                 Part 2: Datatypes";
  }

  typedef date {
    type string {
      pattern '[0-9]{4}-(1[0-2]|0[1-9])-(0[1-9]|[1|2][0-9]|3[0-1])'
            + '(Z|[\+\-]((1[0-3]|0[0-9]):([0-5][0-9])|14:00))?';
    }
    description
     "The date type represents a time-interval of the length
      of a day, i.e., 24 hours.

      The date type is compatible with the XML schema date
      type with the following notable exceptions:

      (a) The date type does not allow negative years.
```

```
      (b) The time-offset -00:00 indicates that the date value is
          reported in UTC and that the local time zone reference point
          is unknown. The time-offsets +00:00 and Z both indicate that
          the date value is reported in UTC and that the local time
          reference point is UTC (see RFC 3339 section 4.3).

      The canonical format for date 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 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 values with an unknown time zone (usually referring
      to the notion of local time) uses the time-offset -00:00,
      i.e., date values must be reported in UTC.";
    reference
     "RFC 3339: Date and Time on the Internet: Timestamps
      XSD-TYPES: XML Schema Definition Language (XSD) 1.1
                 Part 2: Datatypes";
  }

  typedef time {
    type string {
      pattern '(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](\.[0-9]+)?'
            + '(Z|[\+\-]((1[0-3]|0[0-9]):([0-5][0-9])|14:00))?';
    }
    description
     "The time type represents an instance of time of zero-duration
      that recurs every day.

      The time type is compatible with the XML schema time
      type with the following notable exception:

      (a) The time-offset -00:00 indicates that the time value is
          reported in UTC and that the local time zone reference point
          is unknown. The time-offsets +00:00 and Z both indicate that
          the time value is reported in UTC and that the local time
          reference point is UTC (see RFC 3339 section 4.3).

      The canonical format for 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 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
      time values with an unknown time zone (usually referring
```

```
       to the notion of local time) uses the time-offset -00:00,
       i.e., time values must be reported in UTC.";
     reference
      "RFC 3339: Date and Time on the Internet: Timestamps
       XSD-TYPES: XML Schema Definition Language (XSD) 1.1
                  Part 2: Datatypes";
   }

   typedef hours32 {
     type int32;
     units "hours";
     description
      "A period of time, measured in units of hours.

       The maximum time period that can be expressed is in the
       range [-89478485 days 08:00:00 to 89478485 days 07:00:00].

       This type should be range restricted in situations
       where only non-negative time periods are desirable,
       (i.e., range '0..max').";
   }

   typedef minutes32 {
     type int32;
     units "minutes";
     description
      "A period of time, measured in units of minutes.

       The maximum time period that can be expressed is in the
       range [-1491308 days 2:08:00 to 1491308 days 2:07:00].

       This type should be range restricted in situations
       where only non-negative time periods are desirable,
       (i.e., range '0..max').";
   }

   typedef seconds32 {
     type int32;
     units "seconds";
     description
      "A period of time, measured in units of seconds.

       The maximum time period that can be expressed is in the
       range [-24855 days 03:14:08 to 24855 days 03:14:07].

       This type should be range restricted in situations
       where only non-negative time periods are desirable,
       (i.e., range '0..max').";
```

```
  }

  typedef centiseconds32 {
    type int32;
    units "centiseconds";
    description
     "A period of time, measured in units of 10^-2 seconds.

      The maximum time period that can be expressed is in the
      range [-248 days 13:13:56 to 248 days 13:13:56].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  typedef milliseconds32 {
    type int32;
    units "milliseconds";
    description
     "A period of time, measured in units of 10^-3 seconds.

      The maximum time period that can be expressed is in the
      range [-24 days 20:31:23 to 24 days 20:31:23].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  typedef microseconds32 {
    type int32;
    units "microseconds";
    description
     "A period of time, measured in units of 10^-6 seconds.

      The maximum time period that can be expressed is in the
      range [-00:35:47 to 00:35:47].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  typedef microseconds64 {
    type int64;
    units "microseconds";
    description
```

```
     "A period of time, measured in units of 10^-6 seconds.

      The maximum time period that can be expressed is in the
      range [-106751991 days 04:00:54 to 106751991 days 04:00:54].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  typedef nanoseconds32 {
    type int32;
    units "nanoseconds";
    description
     "A period of time, measured in units of 10^-9 seconds.

      The maximum time period that can be expressed is in the
      range [-00:00:02 to 00:00:02].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  typedef nanoseconds64 {
    type int64;
    units "nanoseconds";
    description
     "A period of time, measured in units of 10^-9 seconds.

      The maximum time period that can be expressed is in the
      range [-106753 days 23:12:44 to 106752 days 0:47:16].

      This type should be range restricted in situations
      where only non-negative time periods are desirable,
      (i.e., range '0..max').";
  }

  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 instance 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 schema node instance 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 schema node
      instance 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.";
  }

  /*** collection of YANG specific types ***/

  /* XXX align with YANG 1.1, drop the must not start with xml
     restriction, see email discussion jan '22 */

  typedef yang-identifier {
    type string {
      length "1..max";
      pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
    }
    description
      "A YANG identifier string as defined by the 'identifier'
       rule in Section 14 of RFC 7950. 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.

       This definition conforms to YANG 1.1 defined in RFC
       7950. An earlier version of this definition did exclude
       all identifiers starting with any possible combination
       of the lowercase or uppercase character sequence 'xml',
       as required by YANG 1 defined in RFC 6020. If this type
       is used in a YANG 1 context, then this restriction still
       applies.";
```

```
    reference
      "RFC 7950: The YANG 1.1 Data Modeling Language
       RFC 6020: YANG - A Data Modeling Language for the
                 Network Configuration Protocol (NETCONF)";
  }

  typedef revision-identifier {
    type date {
      pattern '[0-9]{4}-(1[0-2]|0[1-9])-(0[1-9]|[1|2][0-9]|3[0-1])';
    }
    description
     "Represents a specific revision of a YANG module by means of
      a date value without a time zone.";
  }

  typedef percent-i32 {
    type int32;
    units "percent";
    description
     "This type represents a 32-bit signed percentage value.
      Depending on the usage scenario, it may make sense to
      add range constraints. For example, the type definition

          percent-i32 { range '-100..100'; }

      restricts the range to -100 to 100.";
  }

  typedef percent-u32 {
    type uint32;
    units "percent";
    description
     "This type represents a 32-bit unsigned percentage value.
      Depending on the usage scenario, it may make sense to
      add range constraints. For example, the type definition

          percent-u32 { range '0..200'; }

      restricts the range to 0 to 200.";
  }

  typedef percent {
    type uint8;
    units "percent";
    description
     "This type represents an 8-bit unsigned percentage value
      and it is equivalent to the percentage type defined in
      the ietf-routing-types module (RFC 8294). While the
```

```
      type definition

         percent-u32 { range '0..100' }

      yields the same value space, it is possible that encodings
      choose different encodings due to the different base types.";
    reference
      "RFC 8294: Common YANG Data Types for the Routing Area";
  }

}
```

<CODE ENDS>

4. Internet-Specific Derived Types

The ietf-inet-types YANG module references [RFC0768], [RFC0791], [RFC0793], [RFC0952], [RFC1034], [RFC1123], [RFC1930], [RFC2317], [RFC2474], [RFC2780], [RFC2782], [RFC3289], [RFC3305], [RFC3595], [RFC3927], [RFC3986], [RFC4001], [RFC4007], [RFC4271], [RFC4291], [RFC4340], [RFC4592] [RFC4960], [RFC5017], [RFC5322], [RFC5890], [RFC5952], [RFC6793], and [RFC8200].

<CODE BEGINS> file "ietf-inet-types@2022-01-15.yang"

```
 module ietf-inet-types {

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

   organization
    "IETF Network Modeling (NETMOD) Working Group";

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

     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.

     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 (RFC 2119) (RFC 8174) when, and only when,
```

```
       they appear in all capitals, as shown here.

       Copyright (c) 2022 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
       (https://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 2022-01-15 {
       description
        "This revision adds the following new data types:
         - inet:ip-address-and-prefix
         - inet:ipv4-address-and-prefix
         - inet:ipv6-address-and-prefix
         - inet:protocol-number
         - inet:host-name
         - inet:email-address
         - inet:ip-address-link-local
         - inet:ipv4-address-link-local
         - inet:ipv6-address-link-local
         The inet:host union was changed to use inet:host-name instead
         of inet:domain-name.";
       reference
        "RFC XXXX: Common YANG Data Types";
     }

     revision 2013-07-15 {
       description
        "This revision adds the following new data types:
         - inet:ip-address-no-zone
         - inet:ipv4-address-no-zone
         - inet:ipv6-address-no-zone";
       reference
        "RFC 6991: Common YANG Data Types";
     }

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

```
    }

    /*** collection of types related to protocol fields ***/

    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 8200.";
        }
      }
      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 8200: 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 ipv6-flow-label type represents the 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 8200: 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.  The current list of
       all assignments is available from <https://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";
   }

   typedef protocol-number {
```

```
      type uint8;
      description
       "The protocol-number type represents an 8-bit Internet
        protocol number, carried in the 'protocol' field of the
        IPv4 header or in the 'next header' field of the IPv6
        header. If IPv6 extension headers are present, then the
        protocol number type represents the upper layer protocol
        number, i.e., the number of the last next header' field
        of the IPv6 extension headers.

        Protocol numbers are assigned by IANA. The current list of
        all assignments is available from <https://www.iana.org/>.";
      reference
       "RFC  791: Internet Protocol
        RFC 8200: Internet Protocol, Version 6 (IPv6) Specification";
    }

    /*** collection of types related to autonomous systems ***/

    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 ASes.  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 Autonomous System (AS)
                  Number Space";
    }
```

```
    /*** collection of types related to IP addresses and hostnames ***/

    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 textual
        representation defined in Section 4 of RFC 5952.  The
        canonical format for the zone index is the numerical
        format as described in Section 11.2 of RFC 4007.";
      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, derived from
         ipv4-address, 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, derived from
         ipv6-address, 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-address-link-local {
      type union {
        type inet:ipv4-address-link-local;
        type inet:ipv6-address-link-local;
      }
      description
       "The ip-address-link-local type represents a link-local IP
        address and is IP version neutral. The format of the textual
        representation implies the IP version.";
    }

    typedef ipv6-address-link-local {
      type ipv6-address {
        pattern '[fF][eE]80:.*';
      }
      description
        "A link-local IPv6 address in the prefix fe80::/10 as defined
         in section 2.5.6. of RFC 4291.";
      reference
        "RFC 4291: IP Version 6 Addressing Architecture";
    }

    typedef ipv4-address-link-local {
      type ipv4-address {
        pattern '169\.254\..*';
      }
      description
        "A link-local IPv4 address in the prefix 169.254.0.0/16 as
         defined in section 2.1. of RFC 3927.";
      reference
        "RFC 3927: Dynamic Configuration of IPv4 Link-Local Addresses";
    }
```

```
    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 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.

        The definition of ipv4-prefix does not require that bits,
        which are not part of the prefix, are set to zero. However,
        implementations have to return values in canonical format,
        which requires non-prefix bits to be set to zero. This means
        that 192.0.2.1/24 must be accepted as a valid value but it
        will be converted into the canonical format 192.0.2.0/24.";
    }

    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 prefix.
        The prefix length is given by the number following the
        slash character and must be less than or equal to 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 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, the IPv6 address is represented
        as defined in Section 4 of RFC 5952.

        The definition of ipv6-prefix does not require that bits,
        which are not part of the prefix, are set to zero. However,
        implementations have to return values in canonical format,
        which requires non-prefix bits to be set to zero. This means
        that 2001:db8::1/64 must be accepted as a valid value but it
        will be converted into the canonical format 2001:db8::/64.";
      reference
       "RFC 5952: A Recommendation for IPv6 Address Text
                  Representation";
    }

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

    typedef ipv4-address-and-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-address-and-prefix type represents an IPv4
        address and an associated ipv4 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.";
    }

    typedef ipv6-address-and-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-address-and-prefix type represents an IPv6
        address and an associated ipv4 prefix.
        The prefix length is given by the number following the
        slash character and must be less than or equal to 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 canonical format requires that the IPv6 address is
        represented as defined in Section 4 of RFC 5952.";
      reference
       "RFC 5952: A Recommendation for IPv6 Address Text
                  Representation";
    }

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

    typedef domain-name {
      type string {
        length "1..253";
        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]\.?)'
        + '|\.';
      }
      description
       "The domain-name type represents a DNS domain name.  The
```

```
       name SHOULD be fully qualified whenever possible. This
       type does not support wildcards (see RFC 4592) or
       classless in-addr.arpa delegations (see RFC 2317).

       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.  Note that Internet host names have a
       stricter syntax (described in RFC 952) than the DNS
       recommendations in RFCs 1034 and 1123. Schema nodes
       representing host names should use the host-name type
       instead of the domain-type.

       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
       explicitly or 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 A-labels as per RFC 5890.";
     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 2317: Classless IN-ADDR.ARPA delegation
       RFC 2782: A DNS RR for specifying the location of services
                 (DNS SRV)
       RFC 4592: The Role of Wildcards in the Domain Name System
       RFC 5890: Internationalized Domain Names in Applications
                 (IDNA): Definitions and Document Framework";
   }

   typedef host-name {
     type domain-name {
       pattern '[a-zA-Z0-9\-\.]+';
```

```
        length "2..max";
      }
      description
       "The host-name type represents (fully qualified) host names.
        Host names must be at least two characters long (see RFC 952)
        and they are restricted to labels consisting of letters, digits
        and hyphens separated by dots (see RFC1123 and RFC 952).";
      reference
       "RFC  952: DoD Internet Host Table Specification
        RFC 1123: Requirements for Internet Hosts -- Application
                  and Support";
    }

    typedef host {
      type union {
        type inet:ip-address;
        type inet:host-name;
      }
      description
       "The host type represents either an IP address or a (fully
        qualified) host 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)";
    }

    typedef email-address {
      type string {
        pattern '(([a-zA-Z0-9!#$%&'+"'"+'*+/=?\^_`{|}~-]+'
              + '(\.[a-zA-Z0-9!#$%&'+"'"+'*+/=?\^_`{|}~-]+)*)|'
              + '("[a-zA-Z0-9!#$%&'+"'"+'()*+,./\[\]\^_`{|}~-]*"))'
              + '@'
              + '(([a-zA-Z0-9!#$%&'+"'"+'*+/=?\^_`{|}~-]+'
              + '(\.[a-zA-Z0-9!#$%&'+"'"+'*+/=?\^_`{|}~-]+)*)|'
              + '\[[a-zA-Z0-9!"#$%&'+"'"+'()*+,./:;<=>?@\^_`{|}~-]+\])';
      }
      description
        "The email-address type represents an email address as
         defined as addr-spec in RFC 5322 section 3.4.1 except
         that obs-local-part, obs-domain and obs-qtext of the
         quoted-string are not supported.

         The email-address type uses US-ASCII characters. The
         canonical format of the domain part of an email-address
         uses lowercase US-ASCII characters.";
      reference
        "RFC 5322: Internet Message Format";
    }

  }

   <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 XXXX

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

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 [RFC7950] 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, Benoit Claise, Joel M. Halpern, Ladislav Lhotka, Lars-Johan Liman, and Dan Romascanu.

Juergen Schoenwaelder was partly funded by the European Union's Seventh Framework Programme under Grant Agreement ICT-318488 and the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 830927.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <https://www.rfc-editor.org/info/rfc4007>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <https://www.rfc-editor.org/info/rfc8294>.

[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, June 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, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, DOI 10.17487/RFC0952, October 1985, <https://www.rfc-editor.org/info/rfc952>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, DOI 10.17487/RFC1930, March 1996, <https://www.rfc-editor.org/info/rfc1930>.

[RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-ADDR.ARPA delegation", BCP 20, RFC 2317, DOI 10.17487/RFC2317, March 1998, <https://www.rfc-editor.org/info/rfc2317>.

[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, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <https://www.rfc-editor.org/info/rfc2578>.

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, <https://www.rfc-editor.org/info/rfc2579>.

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, DOI 10.17487/RFC2780, March 2000, <https://www.rfc-editor.org/info/rfc2780>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC2856] Bierman, A., McCloghrie, K., and R. Presuhn, "Textual Conventions for Additional High Capacity Data Types", RFC 2856, DOI 10.17487/RFC2856, June 2000, <https://www.rfc-editor.org/info/rfc2856>.

[RFC3289] Baker, F., Chan, K., and A. Smith, "Management Information Base for the Differentiated Services Architecture", RFC 3289, DOI 10.17487/RFC3289, May 2002, <https://www.rfc-editor.org/info/rfc3289>.

[RFC3305] Mealling, M., Ed. and R. Denenberg, Ed., "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, DOI 10.17487/RFC3305, August 2002, <https://www.rfc-editor.org/info/rfc3305>.

[RFC3595] Wijnen, B., "Textual Conventions for IPv6 Flow Label", RFC 3595, DOI 10.17487/RFC3595, September 2003, <https://www.rfc-editor.org/info/rfc3595>.

[RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <https://www.rfc-editor.org/info/rfc3927>.

[RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, DOI 10.17487/RFC4001, February 2005, <https://www.rfc-editor.org/info/rfc4001>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

[RFC4502] Waldbusser, S., "Remote Network Monitoring Management Information Base Version 2", RFC 4502, DOI 10.17487/RFC4502, May 2006, <https://www.rfc-editor.org/info/rfc4502>.

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, DOI 10.17487/RFC4592, July 2006, <https://www.rfc-editor.org/info/rfc4592>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC5017] McWalter, D., Ed., "MIB Textual Conventions for Uniform Resource Identifiers (URIs)", RFC 5017, DOI 10.17487/RFC5017, September 2007, <https://www.rfc-editor.org/info/rfc5017>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <https://www.rfc-editor.org/info/rfc6793>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[XSD-TYPES]
Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C., and H. Thompson, "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, April 2012, <http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>.

Appendix A. Changes from RFC 6991

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

- date, time
- hours32, minutes32, seconds32, centiseconds32, milliseconds32,
- microseconds32, microseconds64, nanoseconds32, nanoseconds64
- revision-identifiers
- percent, percent-i32, percent-u32

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

- ip-address-and-prefix, ipv4-address-and-prefix, ipv6-address-and-prefix
- ip-address-link-local, ipv4-address-link-local, ipv6-address-link-local
- protocol-number
- host-name
- email-address

The yang-identifier definition has been aligned with YANG 1.1. Some pattern statements have been rewritten in order to make them tighter. Finally, this version addresses errata 4076 and 5105 of RFC 6991.

Appendix B. 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:

- o yang-identifier

- o hex-string

- o uuid

- o dotted-quad

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

- o ip-address-no-zone

- o ipv4-address-no-zone

- o ipv6-address-no-zone

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de