Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2009

J. Schoenwaelder, Ed.
Jacobs University

November 3, 2008

Common YANG Data Types
draft-ietf-netmod-yang-types-01

Status of this Memo

Copyright Notice

Abstract

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

Table of Contents

1. Introduction

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

This document introduces a collection of common data types derived from the built-in YANG data types. The definitions are organized in several YANG modules. The "yang-types" module contains generally useful data types. The "inet-types" module contains definitions that are relevant for the Internet protocol suite while the "ieee-types" module contains definitions that are relevant for IEEE 802 protocols.

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

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

2. Core YANG Derived Types

```
module yang-types {

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

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

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

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

    WG Chair: David Harrington
              <mailto:ietfdbh@comcast.net>

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

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

    Copyright (C) The IETF Trust (2008).  This version of this
    YANG module is part of RFC XXXX; see the RFC itself for full
```

```
    legal notices.";
  // RFC Ed.: replace XXXX with actual RFC number and remove this note

  revision 2008-11-03 {
    description
     "Initial revision, published as RFC XXXX.";
  }
  // RFC Ed.: replace XXXX with actual RFC number and remove this note

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

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



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

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

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

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

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

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

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



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

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

```
      an object instance of type counter64 at times other than
      re-initialization, then a corresponding object should be
      defined, with an appropriate type, to indicate the last
      discontinuity.

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

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

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

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

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



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

  typedef gauge32 {
```

```
    type uint32;
    description
     "The gauge32 type represents a non-negative integer, which
      may increase or decrease, but shall never exceed a maximum
      value, nor fall below a minimum value.  The maximum value
      can not be greater than 2^32-1 (4294967295 decimal), and
      the minimum value can not be smaller than 0.  The value of
      a gauge32 has its maximum value whenever the information
      being modeled is greater than or equal to its maximum
      value, and has its minimum value whenever the information
      being modeled is smaller than or equal to its minimum value.
      If the information being modeled subsequently decreases
      below (increases above) the maximum (minimum) value, the
      gauge32 also decreases (increases).

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

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

      This type is in the value set and its semantics equivalent
      to the CounterBasedGauge64 SMIv2 textual convention defined
      in RFC 2856";
```



```
    reference
```

```
     "RFC 2856: Textual Conventions for Additional High Capacity
                Data Types";
  }

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

  typedef object-identifier {
    type string {
      pattern '(([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*))))'
            + '(\.(0|([1-9]\d*)))*';
    }
    description
     "The object-identifier type represents administratively
      assigned names in a registration-hierarchical-name tree.

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

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

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

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

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

```
  }

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

  typedef date-and-time {
    type string {
      pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?'
            + '(Z|(\+|-)\d{2}:\d{2})';
    }
    description
     'The date-and-time type is a profile of the ISO 8601
      standard for representation of dates and times using the
      Gregorian calendar. The format is most easily described
      using the following ABFN (see RFC 3339):

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

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

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

      The date-and-time type is compatible with the dateTime XML
      schema type except that dateTime allows negative years
      which are not allowed by RFC 3339.

      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.';

      // [TODO] This type may require normalization rules since Z and
      // +00:00 mean the same - but note that -00:00 does not according
      // to RFC 3339 section 4.3 but it does according to XSD.
      // In addition, it is possible to write the same data and time
      // value using different time zones. XSD says the canonical format
```

```
      // is UTC using the Z format.
```



```
    reference
     "RFC 3339: Date and Time on the Internet: Timestamps
      RFC 2579: Textual Conventions for SMIv2";
  }

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

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

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

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

      This type is in the value set and its semantics equivalent to
      the TimeStamp textual convention of the SMIv2.";
    reference
```

```
     "RFC 2579: Textual Conventions for SMIv2";
  }

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

  typedef phys-address {
    type string {
      pattern '([0-9a0-fA-F]{2}(:[0-9a0-fA-F]{2})*)?';
    }
```



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

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

}
```

3. Internet Specific Derived Types

```
 module inet-types {

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

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

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

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

     WG Chair: David Harrington
               <mailto:ietfdbh@comcast.net>

     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) The IETF Trust (2008).  This version of this
    YANG module is part of RFC XXXX; see the RFC itself for full
    legal notices.";
  // RFC Ed.: replace XXXX with actual RFC number and remove this note

  revision 2008-11-03 {
    description
     "Initial revision, published as RFC XXXX.";
  }
  // RFC Ed.: replace XXXX with actual RFC number and remove this note

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

  typedef ip-version {
    type enumeration {
      enum unknown {
        value "0";
        description
         "An unknown or unspecified version of the Internet protocol.";
      }
      enum ipv4 {
```



```
        value "1";
        description
         "The IPv4 protocol as defined in RFC 791.";
      }
      enum ipv6 {
        value "2";
        description
         "The IPv6 protocol as defined in RFC 2460.";
      }
    }
    description
     "This value represents the version of the IP protocol.

      This type is in the value set and its semantics equivalent
      to the InetVersion textual convention of the SMIv2. However,
      the lexical appearance is different from the InetVersion
      textual convention.";
    reference
```

```
       "RFC  791: Internet Protocol
        RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
        RFC 4001: Textual Conventions for Internet Network Addresses";
     }

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

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

     typedef flow-label {
       type uint32 {
         range "0..1048575";
       }
       description
        "The flow-label type represents flow identifier or Flow Label
```



```
         in an IPv6 packet header that may be used to discriminate
         traffic flows.

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

     typedef port-number {
```

```
    type uint16 {
      range "1..65535";
    }
    description
     "The port-number type represents a 16-bit port number of an
      Internet transport layer protocol such as UDP, TCP, DCCP or
      SCTP. Port numbers are assigned by IANA.  A current list of
      all assignments is available from <http://www.iana.org/>.

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

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

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

  typedef autonomous-system-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 are currently limited to 16 bits
       (0..65535). There is however work in progress to enlarge
       the autonomous system number space to 32 bits. This
       textual convention therefore uses an uint32 base type
```

```
        without a range restriction in order to support a larger
        autonomous system number space.

        This type is in the value set and its semantics equivalent
        to the InetAutonomousSystemNumber textual convention of
        the SMIv2.";
    reference
     "RFC 1930: Guidelines for creation, selection, and registration
                of an Autonomous System (AS)
      RFC 4271: A Border Gateway Protocol 4 (BGP-4)
      RFC 4001: Textual Conventions for Internet Network Addresses";
  }

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

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

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

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

        // [TODO] There is an normalization issue with regard to
        // systems that allow numeric and textual zone indexes.
    }

    typedef ipv6-address {
      type string {
        pattern
         /* full */
         '((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})'
    +     '(%[\p{N}\p{L}]+)?)'
         /* mixed */
    +    '|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\.'
    +        '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))'
    +      '(%[\p{N}\p{L}]+)?)'
         /* shortened */
    +    '|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)'
    +      '((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*'
    +      '(%[\p{N}\p{L}]+)?)'
         /* shortened mixed */
    +  '|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)'
    +     '((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*'
    +     '(([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))'
    +      '(%[\p{N}\p{L}]+)?)';
      }
      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.";

        // [TODO] Normalization needed due to the shortened and
        // mixed forms and the zone index?

      reference
       "RFC 4007: IPv6 Scoped Address Architecture";
    }
```

```
   // [TODO] The pattern needs to be checked; once YANG supports
   // multiple pattern, we can perhaps be more precise.

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

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

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

       The IPv4 address represented in dotted quad notation
       should have all bits that do not belong to the prefix
       set to zero.";

       // [TODO] Normalization needed since bits of the prefix
       // can be set arbitrarily.
   }

   typedef ipv6-prefix {
     type string {
       pattern
```

```
        /* full */
        '((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})'
      +  '/\d+)'
        /* mixed */
      +  '|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\.'
      +      '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))'
      +   '/\d+)'
        /* shortened */
```



```
      +  '|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)'
      +   '(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*'
      +   '/\d+)'
        /* shortened mixed */
      + '|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)'
      +   '(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*'
      +   '(([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))'
      +    '/\d+)';
    }
    description
     "The ipv6-prefix type represents an IPv6 address prefix.
      The prefix length is given by the number following the
      slash character and must be less than or equal 128.

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

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

      // [TODO] Normalization needed due to the shortened and
      // mixed forms and since bits of the prefix can be set
      // arbitrarily.
  }

  // [TODO] The pattern needs to be checked; once YANG supports
  // multiple pattern, we can perhaps be more precise.]

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

  typedef domain-name {
    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 domain-name type represents a DNS domain name. The
        name SHOULD be fully qualified whenever possible.

        The description clause of objects using the domain-name
        type MUST describe how (and when) these names are
        resolved to IP addresses.

        Note that the resolution of a domain-name value may
        require to query multiple DNS records (e.g., A for IPv4
        and AAAA for IPv6). The order of the resolution process
        and which DNS record takes precedence depends on the
```



```
        configuration of the resolver.";

        // [TODO] Normalization needed since names are case
        // insensitive (normalize to lowercase characters).]

      reference
       "RFC 1034: Domain Names - Concepts and Facilities
        RFC 1123: Requirements for Internet Hosts -- Application
                  and Support";
    }

    // [TODO] RFC 2181 says there are no restrictions on DNS
    // labels. Need to check whether the pattern is too
    // restrictive.

    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;    // [TODO] add the regex from RFC 3986 here?
      description
       "The uri type represents a Uniform Resource Identifier
        (URI) as defined by STD 66.

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

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

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



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

        This type is in the value set and its semantics equivalent
        to the Uri textual convention of the SMIv2.";
      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)";
    }

  }
```

4. IEEE Specific Derived Types

```
  module ieee-types {

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

    import yang-types { prefix yang; }

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

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

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

      WG Chair: David Harrington
                <mailto:ietfdbh@comcast.net>

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

    description
     "This module contains a collection of generally useful derived
      YANG data types for IEEE 802 addresses and related things.

      Copyright (C) The IETF Trust (2008).  This version of this
      YANG module is part of RFC XXXX; see the RFC itself for full
      legal notices.";
    // RFC Ed.: replace XXXX with actual RFC number and remove this note

    revision 2008-11-03 {
      description
       "Initial revision, published as RFC XXXX";
    }
    // RFC Ed.: replace XXXX with actual RFC number and remove this note

    /*** collection of IEEE address type definitions ***/

    typedef mac-address {
      type string {
        pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}';
      }
      description
       "The mac-address type represents an 802 MAC address represented
```



```
        in the `canonical' order defined by IEEE 802.1a, i.e., as if it
        were transmitted least significant bit first, even though 802.5
        (in contrast to other 802.x protocols) requires MAC addresses
        to be transmitted most significant bit first.
```

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

    /*** collection of IEEE 802 related identifier types ***/

    typedef bridgeid {
      type string {
        pattern '[0-9a-fA-F]{4}(:[0-9a-fA-F]{2}){6}';
      }
      description
       "The bridgeid type represents identifiers that uniquely
        identify a bridge.  Its first four hexadecimal digits
        contain a priority value followed by a colon. The
        remaining characters contain the MAC address used to
        refer to a bridge in a unique fashion (typically, the
        numerically smallest MAC address of all ports on the
        bridge).

        This type is in the value set and its semantics equivalent
        to the BridgeId textual convention of the SMIv2. However,
        since the BridgeId textual convention does not prescribe
        a lexical representation, the appearance might be different.";
      reference
       "RFC 4188: Definitions of Managed Objects for Bridges";
    }

    typedef vlanid {
      type uint16 {
        range "1..4094";
      }
      description
       "The vlanid type uniquely identifies a VLAN. This is the
        12-bit VLAN-ID used in the VLAN Tag header. The range is
        defined by the referenced specification.

        This type is in the value set and its semantics equivalent to
        the VlanId textual convention of the SMIv2.";
      reference
       "IEEE Std 802.1Q 2003 Edition: Virtual Bridged Local
                  Area Networks
```

```
        RFC 4363: Definitions of Managed Objects for Bridges with
                  Traffic Classes, Multicast Filtering, and Virtual
                  LAN Extensions";
    }

}
```

5. IANA Considerations

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

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

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

This document registers three URIs1 in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested.

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

Registrant Contact: The NETMOD WG of the IETF.

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

6. Security Considerations

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

7. Contributors

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

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

8. References

8.1. Normative References

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

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

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

[YANG] Bjorklund, M., Ed., "YANG - A data modeling language for NETCONF", draft-ietf-netmod-yang-01 (work in progress).

8.2. Informative References

[802.1Q] ANSI/IEEE Standard 802.1Q, "IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area

Networks", 2003.

[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.

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

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

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

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

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

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

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

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

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For

Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.

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

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

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

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

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

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

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

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

Addresses", RFC 4001, February 2005.

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

[RFC4188] Norseth, K. and E. Bell, "Definitions of Managed Objects

for Bridges", RFC 4188, September 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.

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

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

Appendix A. XSD Translations

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

A.1. XSD of Core YANG Derived Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:yang:yang-types"
           xmlns="urn:ietf:params:xml:ns:yang:yang-types"
           xmlns:yang="urn:ietf:params:xml:ns:yang:yang-types"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           version="2008-11-03"
           xml:lang="en">

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

      Copyright (C) The IETF Trust (2008).  This version of this
      YANG module is part of RFC XXXX; see the RFC itself for full
      legal notices.
    </xs:documentation>
  </xs:annotation>

  <!-- YANG typedefs -->

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

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

```
        defined, with an appropriate type, to indicate the last
        discontinuity.

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

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

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

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

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

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

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

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

```
  <xs:simpleType name="counter64">
    <xs:annotation>
      <xs:documentation>
        The counter64 type represents a non-negative integer
        which monotonically increases until it reaches a
```



```
        maximum value of 2^64-1 (18446744073709551615), when
        it wraps around and starts increasing again from zero.

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

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

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

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

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

        Objects of this type will be set to zero(0) on creation
        and will thereafter count appropriate events, wrapping
```

back to zero(0) when the value 2^64 is reached.

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

This type is in the value set and its semantics equivalent to the ZeroBasedCounter64 textual convention of the SMIv2.


```
      </xs:documentation>
    </xs:annotation>

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

  <xs:simpleType name="gauge32">
    <xs:annotation>
      <xs:documentation>
        The gauge32 type represents a non-negative integer, which
        may increase or decrease, but shall never exceed a maximum
        value, nor fall below a minimum value.  The maximum value
        can not be greater than 2^32-1 (4294967295 decimal), and
        the minimum value can not be smaller than 0.  The value of
        a gauge32 has its maximum value whenever the information
        being modeled is greater than or equal to its maximum
        value, and has its minimum value whenever the information
        being modeled is smaller than or equal to its minimum value.
        If the information being modeled subsequently decreases
        below (increases above) the maximum (minimum) value, the
        gauge32 also decreases (increases).

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

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

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

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



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

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

  <xs:simpleType name="object-identifier">
    <xs:annotation>
      <xs:documentation>
        The object-identifier type represents administratively
        assigned names in a registration-hierarchical-name tree.

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

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

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

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

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

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



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

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

        date-fullyear   = 4DIGIT
        date-month      = 2DIGIT  ; 01-12
```

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

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

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

      The date-and-time type is compatible with the dateTime XML
      schema type except that dateTime allows negative years
      which are not allowed by RFC 3339.

      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.
    </xs:documentation>
  </xs:annotation>

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

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



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

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

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

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

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

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

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

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



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

```
        the PhysAddress textual convention of the SMIv2.
      </xs:documentation>
    </xs:annotation>

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


</xs:schema>
```

A.2. XSD of Internet Specific Derived Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:yang:inet-types"
           xmlns="urn:ietf:params:xml:ns:yang:inet-types"
           xmlns:inet="urn:ietf:params:xml:ns:yang:inet-types"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           version="2008-11-03"
           xml:lang="en">

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

      Copyright (C) The IETF Trust (2008).  This version of this
      YANG module is part of RFC XXXX; see the RFC itself for full
      legal notices.
    </xs:documentation>
  </xs:annotation>

  <!-- YANG typedefs -->

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

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

```
      </xs:annotation>

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

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

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

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

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

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

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

    <xs:simpleType name="port-number">
      <xs:annotation>
```

```
      <xs:documentation>
        The port-number type represents a 16-bit port number of an
```



```
        Internet transport layer protocol such as UDP, TCP, DCCP or
        SCTP. Port numbers are assigned by IANA.  A current list of
        all assignments is available from &lt;http://www.iana.org/&gt;.

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

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

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

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

        Autonomous system numbers are currently limited to 16 bits
        (0..65535). There is however work in progress to enlarge
        the autonomous system number space to 32 bits. This
        textual convention therefore uses an uint32 base type
        without a range restriction in order to support a larger
        autonomous system number space.

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

```
        to the InetAutonomousSystemNumber textual convention of
        the SMIv2.
      </xs:documentation>
    </xs:annotation>

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



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

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

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

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

```
        zone of the device will be used.
      </xs:documentation>
    </xs:annotation>

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

  <xs:simpleType name="ipv6-address">
    <xs:annotation>
      <xs:documentation>
        The ipv6-address type represents an IPv6 address in full,
```



```
        mixed, shortened and shortened mixed notation.  The IPv6
        address may include a zone index, separated by a % sign.

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

    <xs:restriction base="xs:string">
      <xs:pattern value="((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})
                        (%[\p{N}\p{L}]+)?)|((([0-9a-fA-F]{1,4}:){6})
                        (([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{
                        1,3}))(%[\p{N}\p{L}]+)?)|((([0-9a-fA-F]{1,4}
                        :)*([0-9a-fA-F]{1,4}))*(::)(([0-9a-fA-F]{1,4
                        }:)*([0-9a-fA-F]{1,4}))*(%[\p{N}\p{L}]+)?)|(
                        (([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::
                        )(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*((
                        [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,
                        3}))(%[\p{N}\p{L}]+)?)"/>
    </xs:restriction>
  </xs:simpleType>
```

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

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

   <xs:simpleType name="ipv4-prefix">
     <xs:annotation>
```



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

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

         The IPv4 address represented in dotted quad notation
         should have all bits that do not belong to the prefix
         set to zero.
       </xs:documentation>
     </xs:annotation>

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

```
                        d+"/>
    </xs:restriction>
  </xs:simpleType>

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

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

        The IPv6 address should have all bits that do not belong
        to the prefix set to zero.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})
                         /\d+)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\
                         .[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))/\d+)|(
                         (([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::
                         )(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*/\
                         d+)|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4})
                         )*(::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}
                         ))*(([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
                         9]{1,3}))/\d+)"/>
```



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

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

        The description clause of objects using the domain-name
        type MUST describe how (and when) these names are
```

```
          resolved to IP addresses.

          Note that the resolution of a domain-name value may
          require to query multiple DNS records (e.g., A for IPv4
          and AAAA for IPv6). The order of the resolution process
          and which DNS record takes precedence depends on the
          configuration of the resolver.
        </xs:documentation>
      </xs:annotation>

      <xs:restriction base="xs:string">
        <xs:pattern value="([a-zA-Z0-9][a-zA-Z0-9\-]*[a-zA-Z0-9]\.)*[a
                           -zA-Z0-9][a-zA-Z0-9\-]*[a-zA-Z0-9]"/>
      </xs:restriction>
    </xs:simpleType>

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

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

    <xs:simpleType name="uri">
```



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

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

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

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

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

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

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


 </xs:schema>
```

A.3. XSD of IEEE Specific Derived Types

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:ietf:params:xml:ns:yang:ieee-types"
            xmlns="urn:ietf:params:xml:ns:yang:ieee-types"
            xmlns:ieee="urn:ietf:params:xml:ns:yang:ieee-types"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="2008-11-03"
```

```
            xml:lang="en"
            xmlns:yang="urn:ietf:params:xml:ns:yang:yang-types">

   <xs:import namespace="urn:ietf:params:xml:ns:yang:yang-types"
              schemaLocation="yang-types.xsd"/>

   <xs:annotation>
     <xs:documentation>
       This module contains a collection of generally useful derived
       YANG data types for IEEE 802 addresses and related things.

       Copyright (C) The IETF Trust (2008).  This version of this
       YANG module is part of RFC XXXX; see the RFC itself for full
       legal notices.
     </xs:documentation>
   </xs:annotation>

   <!-- YANG typedefs -->

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

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

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

   <xs:simpleType name="bridgeid">
     <xs:annotation>
       <xs:documentation>
         The bridgeid type represents identifiers that uniquely
         identify a bridge.  Its first four hexadecimal digits
         contain a priority value followed by a colon. The
         remaining characters contain the MAC address used to
         refer to a bridge in a unique fashion (typically, the
         numerically smallest MAC address of all ports on the
```

bridge).


```
        This type is in the value set and its semantics equivalent
        to the BridgeId textual convention of the SMIv2. However,
        since the BridgeId textual convention does not prescribe
        a lexical representation, the appearance might be different.
      </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9a-fA-F]{4}(:[0-9a-fA-F]{2}){6}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="vlanid">
    <xs:annotation>
      <xs:documentation>
        The vlanid type uniquely identifies a VLAN. This is the
        12-bit VLAN-ID used in the VLAN Tag header. The range is
        defined by the referenced specification.

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

    <xs:restriction base="xs:unsignedShort">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="4094"/>
    </xs:restriction>
  </xs:simpleType>


</xs:schema>
```

Appendix B. RelaxNG Translations

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

B.1. RelaxNG of Core YANG Derived Types

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

dc:creator [
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
]
dc:description [
  "This module contains a collection of generally useful derived\x{a}" ~
  "YANG data types.\x{a}" ~
  "\x{a}" ~
  "Copyright (C) The IETF Trust (2008). This version of this\x{a}" ~
  "YANG module is part of RFC XXXX; see the RFC itself for full\x{a}" ~
  "legal notices."
]
dc:issued [ "2008-11-03" ]
dc:source [ "YANG module 'yang-types' (automatic translation)" ]
dc:contributor [
  "WG Web:   <http://tools.ietf.org/wg/netmod/>\x{a}" ~
  "WG List:  <mailto:netmod@ietf.org>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Partain\x{a}" ~
  "          <mailto:david.partain@ericsson.com>\x{a}" ~
```

```
  "\x{a}" ~
  "WG Chair: David Harrington\x{a}" ~
  "          <mailto:ietfdbh@comcast.net>\x{a}" ~
  "\x{a}" ~
  "Editor:   Juergen Schoenwaelder\x{a}" ~
  "          <mailto:j.schoenwaelder@jacobs-university.de>"
]

## The counter32 type represents a non-negative integer
## which monotonically increases until it reaches a
## maximum value of 2^32-1 (4294967295 decimal), when it
## wraps around and starts increasing again from zero.
##
## Counters have no defined `initial' value, and thus, a
## single value of a counter has (in general) no information
```



```
## content.  Discontinuities in the monotonically increasing
## value normally occur at re-initialization of the
## management system, and at other times as specified in the
## description of an object instance using this type.  If
## such other times can occur, for example, the creation of
## an object instance of type counter32 at times other than
## re-initialization, then a corresponding object should be
## defined, with an appropriate type, to indicate the last
## discontinuity.
##
## The counter32 type should not be used for configuration
## objects. A default statement should not be used for
## attributes with a type value of counter32.
##
## This type is in the value set and its semantics equivalent
## to the Counter32 type of the SMIv2.

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

## The zero-based-counter32 type represents a counter32
## which has the defined `initial' value zero.
##
## Objects of this type will be set to zero(0) on creation
## and will thereafter count appropriate events, wrapping
## back to zero(0) when the value 2^32 is reached.
```

```
##
## Provided that an application discovers the new object within
## the minimum time to wrap it can use the initial value as a
## delta since it last polled the table of which this object is
## part.  It is important for a management station to be aware
## of this minimum time and the actual time between polls, and
## to discard data if the actual time is too long or there is
## no defined minimum time.
##
## This type is in the value set and its semantics equivalent
## to the ZeroBasedCounter32 textual convention of the SMIv2.

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

## The counter64 type represents a non-negative integer
## which monotonically increases until it reaches a
## maximum value of 2^64-1 (18446744073709551615), when
## it wraps around and starts increasing again from zero.
##
## Counters have no defined `initial' value, and thus, a
```



```
## single value of a counter has (in general) no information
## content.  Discontinuities in the monotonically increasing
## value normally occur at re-initialization of the
## management system, and at other times as specified in the
## description of an object instance using this type.  If
## such other times can occur, for example, the creation of
## an object instance of type counter64 at times other than
## re-initialization, then a corresponding object should be
## defined, with an appropriate type, to indicate the last
## discontinuity.
##
## The counter64 type should not be used for configuration
## objects. A default statement should not be used for
## attributes with a type value of counter64.
##
## This type is in the value set and its semantics equivalent
## to the Counter64 type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
```

```
counter64 = xsd:unsignedLong

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

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

## The gauge32 type represents a non-negative integer, which
## may increase or decrease, but shall never exceed a maximum
## value, nor fall below a minimum value.  The maximum value
## can not be greater than 2^32-1 (4294967295 decimal), and
## the minimum value can not be smaller than 0.  The value of
```



```
## a gauge32 has its maximum value whenever the information
## being modeled is greater than or equal to its maximum
## value, and has its minimum value whenever the information
## being modeled is smaller than or equal to its minimum value.
## If the information being modeled subsequently decreases
## below (increases above) the maximum (minimum) value, the
## gauge32 also decreases (increases).
##
## This type is in the value set and its semantics equivalent
## to the Counter32 type of the SMIv2.

## See: RFC 2578: Structure of Management Information Version 2 (SMIv2)
```

```
gauge32 = xsd:unsignedInt

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

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

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



```
##
## This type is a superset of the SMIv2 OBJECT IDENTIFIER type
## since it is not restricted to 128 sub-identifiers.

## See: ISO/IEC 9834-1: Information technology -- Open Systems
```

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

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

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

## The date-and-time type is a profile of the ISO 8601
##       standard for representation of dates and times using the
##       Gregorian calendar. The format is most easily described
##       using the following ABFN (see RFC 3339):
##
##       date-fullyear   = 4DIGIT
##       date-month      = 2DIGIT  ; 01-12
##       date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31
##       time-hour       = 2DIGIT  ; 00-23
##       time-minute     = 2DIGIT  ; 00-59
##       time-second     = 2DIGIT  ; 00-58, 00-59, 00-60
##       time-secfrac    = "." 1*DIGIT
##       time-numoffset  = ("+" / "-") time-hour ":" time-minute
##       time-offset     = "Z" / time-numoffset
##
##       partial-time    = time-hour ":" time-minute ":" time-second
##                         [time-secfrac]
##       full-date       = date-fullyear "-" date-month "-" date-mday
##       full-time       = partial-time time-offset
##
##       date-time       = full-date "T" full-time
##
##       The date-and-time type is compatible with the dateTime XML
##       schema type except that dateTime allows negative years
##       which are not allowed by RFC 3339.
```

```
##
##       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.

## See: RFC 3339: Date and Time on the Internet: Timestamps
## RFC 2579: Textual Conventions for SMIv2
date-and-time =
  xsd:string {
    pattern =
      "\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|(\+|-)\d{2}:\d{2})"
  }

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

## See: RFC 2579: Textual Conventions for SMIv2
timeticks = xsd:unsignedInt

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

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

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

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

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

B.2. RelaxNG of Internet Specific Derived Types

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

dc:creator [
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
]
dc:description [
  "This module contains a collection of generally useful derived\x{a}" ~
  "YANG data types for Internet addresses and related things.\x{a}" ~
  "\x{a}" ~
  "Copyright (C) The IETF Trust (2008).  This version of this\x{a}" ~
  "YANG module is part of RFC XXXX; see the RFC itself for full\x{a}" ~
  "legal notices."
]
dc:issued [ "2008-11-03" ]
dc:source [ "YANG module 'inet-types' (automatic translation)" ]
dc:contributor [
  "WG Web:   <http://tools.ietf.org/wg/netmod/>\x{a}" ~
  "WG List:  <mailto:netmod@ietf.org>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Partain\x{a}" ~
  "          <mailto:david.partain@ericsson.com>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Harrington\x{a}" ~
  "          <mailto:ietfdbh@comcast.net>\x{a}" ~
  "\x{a}" ~
  "Editor:   Juergen Schoenwaelder\x{a}" ~
  "          <mailto:j.schoenwaelder@jacobs-university.de>"
```

```
]

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



```
## the lexical appearance is different from the InetVersion
## textual convention.

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

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

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

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

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

## The port-number type represents a 16-bit port number of an
```

```
## Internet transport layer protocol such as UDP, TCP, DCCP or
## SCTP. Port numbers are assigned by IANA.  A current list of
## all assignments is available from <http://www.iana.org/>.
##
## Note that the value zero is not a valid port number. A union
## type might be used in situations where the value zero is
## meaningful.
##
## This type is in the value set and its semantics equivalent
## to the InetPortNumber textual convention of the SMIv2.

## See: RFC  768: User Datagram Protocol
## RFC  793: Transmission Control Protocol
```



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

## The as-number type represents autonomous system numbers
## which identify an Autonomous System (AS). An AS is a set
## of routers under a single technical administration, using
## an interior gateway protocol and common metrics to route
## packets within the AS, and using an exterior gateway
## protocol to route packets to other ASs'. IANA maintains
## the AS number space and has delegated large parts to the
## regional registries.
##
## Autonomous system numbers are currently limited to 16 bits
## (0..65535). There is however work in progress to enlarge
## the autonomous system number space to 32 bits. This
## textual convention therefore uses an uint32 base type
## without a range restriction in order to support a larger
## autonomous system number space.
##
## This type is in the value set and its semantics equivalent
## to the InetAutonomousSystemNumber textual convention of
## the SMIv2.

## See: RFC 1930: Guidelines for creation, selection, and registration
##           of an Autonomous System (AS)
```

```
## RFC 4271: A Border Gateway Protocol 4 (BGP-4)
## RFC 4001: Textual Conventions for Internet Network Addresses
autonomous-system-number = xsd:unsignedInt

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

## The ipv4-address type represents an IPv4 address in
## dotted-quad notation. The IPv4 address may include a zone
## index, separated by a % sign.
##
## The zone index is used to disambiguate identical address
## values.  For link-local addresses, the zone index will
## typically be the interface index number or the name of an
## interface. If the zone index is not present, the default
## zone of the device will be used.
ipv4-address =
  xsd:string {
```



```
    pattern =
      "((0|(1[0-9]{0,2})|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)"
    ~ "))|([3-9][0-9]?))\.){3}(0|(1[0-9]{0,2})|(2(([0-4][0-9]?)|(5["
    ~ "0-5]?)|([6-9]?)))|([3-9][0-9]?))(%[\p{N}\p{L}]+)?"
  }

## The ipv6-address type represents an IPv6 address in full,
## mixed, shortened and shortened mixed notation.  The IPv6
## address may include a zone index, separated by a % sign.
##
## The zone index is used to disambiguate identical address
## values.  For link-local addresses, the zone index will
## typically be the interface index number or the name of an
## interface. If the zone index is not present, the default
## zone of the device will be used.

## See: RFC 4007: IPv6 Scoped Address Architecture
ipv6-address =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})(%[\p{N}\p"
```

```
    ~ "{L}]+)?)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\.[0-9]{1,3}\."
    ~ "[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]+)?)|((([0-9a-fA-F]{1,"
    ~ "4}:)*([0-9a-fA-F]{1,4}))*(::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-"
    ~ "F]{1,4}))*(%[\p{N}\p{L}]+)?)|((([0-9a-fA-F]{1,4}:)*([0-9a-fA"
    ~ "-F]{1,4}))*(::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(([0"
    ~ "-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]"
    ~ "+)?)"
  }

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

## The ipv4-prefix type represents an IPv4 address prefix.
## The prefix length is given by the number following the
## slash character and must be less than or equal to 32.
##
## A prefix length value of n corresponds to an IP address
## mask which has n contiguous 1-bits from the most
## significant bit (MSB) and all other bits set to 0.
##
## The IPv4 address represented in dotted quad notation
## should have all bits that do not belong to the prefix
## set to zero.
ipv4-prefix =
  xsd:string {
```



```
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-1]?"
    ~ "[0-9]?[0-9]|2[0-4][0-9]|25[0-5])/\d+"
  }

## The ipv6-prefix type represents an IPv6 address prefix.
## The prefix length is given by the number following the
## slash character and must be less than or equal 128.
##
## A prefix length value of n corresponds to an IP address
## mask which has n contiguous 1-bits from the most
## significant bit (MSB) and all other bits set to 0.
##
## The IPv6 address should have all bits that do not belong
```

```
## to the prefix set to zero.
ipv6-prefix =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})/\d+)|((([0"
    ~ "-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.["
    ~ "0-9]{1,3}))/\d+)|((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*("
    ~ "::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*/\d+)|((([0-9a-f"
    ~ "A-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)(([0-9a-fA-F]{1,4}:)*([0"
    ~ "-9a-fA-F]{1,4}))*(([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]"
    ~ "{1,3}))/\d+)"
  }

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

## See: RFC 1034: Domain Names - Concepts and Facilities
## RFC 1123: Requirements for Internet Hosts -- Application
##           and Support
domain-name =
  xsd: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]"
```



```
  }

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

## The uri type represents a Uniform Resource Identifier
```

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

## See: RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
## RFC 3305: Report from the Joint W3C/IETF URI Planning Interest
##           Group: Uniform Resource Identifiers (URIs), URLs,
##           and Uniform Resource Names (URNs): Clarifications
##           and Recommendations
## RFC 5017: MIB Textual Conventions for Uniform Resource
##           Identifiers (URIs)
uri = xsd:string
```

B.3. RelaxNG of IEEE Specific Derived Types

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

```
namespace nm = "urn:ietf:params:xml:ns:netmod:dsdl-attrib:1"
namespace sch = "http://purl.oclc.org/dsdl/schematron"

dc:creator [
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
]
dc:description [
  "This module contains a collection of generally useful derived\x{a}" ~
  "YANG data types for IEEE 802 addresses and related things.\x{a}" ~
  "\x{a}" ~
  "Copyright (C) The IETF Trust (2008).  This version of this\x{a}" ~
  "YANG module is part of RFC XXXX; see the RFC itself for full\x{a}" ~
  "legal notices."
]
dc:issued [ "2008-11-03" ]
dc:source [ "YANG module 'ieee-types' (automatic translation)" ]
dc:contributor [
  "WG Web:   <http://tools.ietf.org/wg/netmod/>\x{a}" ~
  "WG List:  <mailto:netmod@ietf.org>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Partain\x{a}" ~
  "          <mailto:david.partain@ericsson.com>\x{a}" ~
  "\x{a}" ~
  "WG Chair: David Harrington\x{a}" ~
  "          <mailto:ietfdbh@comcast.net>\x{a}" ~
  "\x{a}" ~
  "Editor:   Juergen Schoenwaelder\x{a}" ~
  "          <mailto:j.schoenwaelder@jacobs-university.de>"
]

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

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

## The bridgeid type represents identifiers that uniquely
## identify a bridge.  Its first four hexadecimal digits
## contain a priority value followed by a colon. The
## remaining characters contain the MAC address used to
## refer to a bridge in a unique fashion (typically, the
```

```
## numerically smallest MAC address of all ports on the
## bridge).
##
## This type is in the value set and its semantics equivalent
## to the BridgeId textual convention of the SMIv2. However,
## since the BridgeId textual convention does not prescribe
## a lexical representation, the appearance might be different.

## See: RFC 4188: Definitions of Managed Objects for Bridges
bridgeid = xsd:string { pattern = "[0-9a-fA-F]{4}(:[0-9a-fA-F]{2}){6}" }

## The vlanid type uniquely identifies a VLAN. This is the
## 12-bit VLAN-ID used in the VLAN Tag header. The range is
## defined by the referenced specification.
##
## This type is in the value set and its semantics equivalent to
## the VlanId textual convention of the SMIv2.

## See: IEEE Std 802.1Q 2003 Edition: Virtual Bridged Local
##           Area Networks
## RFC 4363: Definitions of Managed Objects for Bridges with
##           Traffic Classes, Multicast Filtering, and Virtual
##           LAN Extensions
vlanid = xsd:unsignedShort { minInclusive = "1" maxInclusive = "4094" }
```

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Full Copyright Statement

Intellectual Property

assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).