Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2015

A. Choudhary
S. Shah
Cisco Systems
M. Jethanandani
Ciena Corporation
Y. Yan
B. Liu
Huawei Technologies
N. Strahle
Juniper Networks
March 04, 2015

# YANG Model for Diffserv
# draft-asechoud-netmod-diffserv-model-01

## Abstract

This document describes a YANG model of Differentiated Services for configuration and operations.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 5, 2015.

## Copyright Notice

Table of Contents



## 1. Introduction

This document defines a YANG [RFC6020] data model for the configuration, state data of Differentiated Services. Any RPC or notification definition is not part of this document. As many vendors have different object constructs to represent the same data, it has been tried to design this model in a very flexible, extensible and generic way to fit into most of the vendor requirements. The model is based on Differentiated Services (Diffserv) architecture and various references have been made to already available standard architecture documents.

Diffserv is a preferred approach for network service providers to offer services to different customers based on their different kinds of network quality-of-service (QoS) objectives. The traffic streams are differentiated based on Differentiated Services Code Points (DSCP) carried in the IP header of each packet. The DSCP markings are applied by upstream node or by the edge router on entry to the Diffserv network.

## 2. Terminology

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

## 3. Diffserv Model Design

Diffserv architecture [RFC3289] [RFC2475] describes network node packet classification function and packet conditioning functions.

The complex classification is done at the edge of network and non-edge network devices conditions appropriately marked aggregate traffic based on per-hop behavior rules. Accordingly, a Multi-Field classifier matches the different fields in a packet and a Behavior Aggregated Classifier matches on DS codepoint field of a packet.

Packets MAY be grouped when a logical set of rules are applied on different packet header fields. Also, packet grouping MAY be done based on different values or range of values of same packet header field. Packet grouping MAY also be done based on presence of some values or range of values of a packet field or absence of such values or ranges. This diffserv model is flexible enough to support such logical grouping of packets.

A classifier entry can be stored as an object and used across different interfaces for either of inbound or outbound traffic. Any modification or deletion of such object will in turn results in such changes to the classifier on the corresponding interfaces. A classifier entry contains one or more packet conditioning functions. A packet conditioning function is typically based on direction of traffic and may drop, mark or delay network packets. A set of such classifier entries with corresponding conditioning functions when arranged in order of priority represents a diffserv policy. Any new classifier entry in a policy MAY be inserted before or after any other existing classifier-entry [RFC6020]. Such policies is stored as an object and used across different network device interfaces.

A meter qualifies if the traffic arrival rate is based on agreed upon rate and variability. A meter is generically modeled as qualifying rate and variability defined as a token bucket. Single rate meter [RFC2697] can be defined as two such token buckets with first defining the rate and committed burst and excess burst for second bucket. Similarly, two rates meter [RFC2698][RFC2859] can be defined as two such token buckets with first and second defining the committed rate and committed burst parameters and peak rate and peak burst respectively. Different Vendors can extend it to have other types of meters as well.

Metered traffic to each token bucket MAY either be marked or remarked appropriately of the diffserv codepoint packet field or even MAY be dropped. Classified packets through a classifier entry MAY directly be marked.

Packets can be always dropped if exceed agreed upon rates or it could be queued and then dropped based on any of various algorithms. Queue dropping is based on the threshold configured and can head-drop, tail-drop or dropped based on Active Queue Management algorithm like Random Early Detection (RED). Packets can be scheduled out based on priority with minimum-rate or WFQ with bandwidth sharing. Priority scheduler allow queue to use the entire capacity of the interface unless higher priority traffic is queued to be scheduled. If combination of EF [RFC3246] and multiple AF [RFC3260] classes of traffic needs to be scheduled, a combination of priority and WFQ scheduler SHOULD be used. Traffic can be shaped by defining a max rate and burst for a leaky bucket profile.

## 4. Diffserv Model

The model have four YANG modules. ietf-diffserv-classifier consists of classifier entries identified by a classifier entry name. Each such entry contains list of filter entries. Each filter entry represent any of the filter type [RFC6991] of a multi-field classifier which can be logically AND/OR with other filter types in the same classifier-entry. The model is flexible enough to take multiple values of the same filter type.

```
module: ietf-diffserv-classifier
 +--rw classifiers
    +--rw classifier-entry* [classifier-entry-name]
       +--rw classifier-entry-name                string
       +--rw classifier-entry-descr?              string
       +--rw classifier-entry-filter-operation?   identityref
       +--rw filter-entry* [filter-type filter-logical-not]
          +--rw filter-type                   identityref
          +--rw filter-logical-not            boolean
          +--rw (filter-param)?
             +--:(dscp)
             |  +--rw dscp-cfg* [dscp-min dscp-max]
             |     +--rw dscp-min    inet:dscp
             |     +--rw dscp-max    inet:dscp
             +--:(source-ip-address)
             |  +--rw source-ip-address-cfg* [source-ip-addr]
             |     +--rw source-ip-addr    inet:ip-prefix
             +--:(destination-ip-address)
             |  +--rw destination-ip-address-cfg*
                         [destination-ip-addr]
             |     +--rw destination-ip-addr    inet:ip-prefix
             +--:(source-port)
             |  +--rw source-port-cfg*
                         [source-port-min source-port-max]
             |     +--rw source-port-min    inet:port-number
             |     +--rw source-port-max    inet:port-number
             +--:(destination-port)
             |  +--rw destination-port-cfg*
                         [destination-port-min destination-port-max]
             |     +--rw destination-port-min    inet:port-number
             |     +--rw destination-port-max    inet:port-number
             +--:(protocol)
             |  +--rw protocol-cfg* [protocol-min protocol-max]
             |     +--rw protocol-min    uint8
             |     +--rw protocol-max    uint8
             +--:(flow-label)
                +--rw flow-label-cfg* [flow-label-min flow-label-max]
                   +--rw flow-label-min    uint32
                   +--rw flow-label-max    uint32
```

An ietf-diffserv-policy module contains list of policy objects identified by a policy name which MUST be provided. Each policy object contains list of classifier-entries either configured inline or referred as an object. Each such classifier entry is augmented by set of actions. A policy object MAY contain a child-policy in each classifier-entry. A child policy MAY further classify the traffic and execute actions on classified packets.

```
module: ietf-diffserv-policy
+--rw policies {policy-template-support}?
   +--rw policy-entry* [policy-name]
      +--rw policy-name         string
      +--rw policy-descr?       string
      +--rw classifier-entry* [classifier-entry-name]
         +--rw classifier-entry-name           string
         +--rw classifier-entry-inline?        boolean
         +--rw classifier-entry-filter-oper?   identityref
         +--rw filter-entry* [filter-type filter-logical-not]
                                 {policy-inline-classifier-config}?
         |  +--rw filter-type                   identityref
         |  +--rw filter-logical-not            boolean
         |  +--rw (filter-param)?
         |     +--:(dscp)
         |     |  +--rw dscp-cfg* [dscp-min dscp-max]
         |     |     +--rw dscp-min    inet:dscp
         |     |     +--rw dscp-max    inet:dscp
         |     +--:(source-ip-address)
         |     |  +--rw source-ip-address-cfg* [source-ip-addr]
         |     |     +--rw source-ip-addr    inet:ip-prefix
         |     +--:(destination-ip-address)
         |     |  +--rw destination-ip-address-cfg*
                                 [destination-ip-addr]
         |     |     +--rw destination-ip-addr    inet:ip-prefix
         |     +--:(source-port)
         |     |  +--rw source-port-cfg*
                                 [source-port-min source-port-max]
         |     |     +--rw source-port-min    inet:port-number
         |     |     +--rw source-port-max    inet:port-number
         |     +--:(destination-port)
         |     |  +--rw destination-port-cfg*
                        [destination-port-min destination-port-max]
         |     |     +--rw destination-port-min    inet:port-number
         |     |     +--rw destination-port-max    inet:port-number
         |     +--:(protocol)
         |     |  +--rw protocol-cfg* [protocol-min protocol-max]
         |     |     +--rw protocol-min    uint8
         |     |     +--rw protocol-max    uint8
         |     +--:(flow-label)
         |        +--rw flow-label-cfg*
                                 [flow-label-min flow-label-max]
         |           +--rw flow-label-min    uint32
         |           +--rw flow-label-max    uint32
         +--rw classifier-action-entry-cfg* [action-type]
         |  +--rw action-type                          identityref
         |  +--rw (action-cfg-params)?
         |     +--:(marking)
```

```
             |     |  +--rw diffserv-action:marking-cfg
             |     |     +--rw diffserv-action:dscp?   inet:dscp
             |     +--:(priority)
             |     |  +--rw diffserv-action:priority-cfg
             |     |     +--rw diffserv-action:priority-level?   uint8
             |     |     +--rw diffserv-action:priority-rate?    uint64
             |     +--:(meter)
             |     |  +--rw diffserv-action:meter-cfg
             |     |     +--rw diffserv-action:meter-list* [meter-id]
             |     |        +--rw diffserv-action:meter-id       uint16
             |     |        +--rw diffserv-action:meter-rate?    uint64
             |     |        +--rw (burst-type)?
             |     |        |  +--:(size)
             |     |        |  |  +--rw diffserv-action:burst-size?
                                                                 uint64
             |     |        |  +--:(interval)
             |     |        |     +--rw diffserv-action:burst-interval?
                                                                 uint64
             |     |        +--rw diffserv-action:color
             |     |        |  +--rw diffserv-action:
                                       classifier-entry-name?  string
             |     |        |  +--rw diffserv-action:
                                       classifier-entry-descr? string
             |     |        |  +--rw diffserv-action:
                                     classifier-entry-filter-operation?
                                                           identityref
             |     |        +--rw diffserv-action:meter-action-type?
                                                           identityref
             |     |        +--rw (val)?
             |     |           +--:(meter-action-mark)
             |     |           |  +--rw diffserv-action:dscp?  inet:dscp
             |     |           +--:(meter-action-drop)
             |     |              +--rw diffserv-action:drop-action?
                                                                 boolean
             |     +--:(max-rate)
             |     |  +--rw diffserv-action:max-rate-cfg
             |     |     +--rw diffserv-action:absolute-rate?    uint64
             |     |     +--rw (burst-type)?
             |     |        +--:(size)
             |     |        |  +--rw diffserv-action:burst-size? uint64
             |     |        +--:(interval)
             |     |           +--rw diffserv-action:burst-interval?
                                                                 uint64
             |     +--:(algorithmic-drop)
             |     |  +--rw (drop-algorithm)?
             |     |     +--:(always-drop)
             |     |     |  +--rw diffserv-action:drop-cfg
             |     |     |     +--rw diffserv-action:drop-action? boolean
```

```
             |     |     +--:(tail-drop)
             |     |     |  +--rw diffserv-action:tail-drop-cfg
             |     |     |     +--rw diffserv-action:qlimit-dscp-thresh*
                                                     [dscp-min dscp-max]
             |     |     |        +--rw diffserv-action:dscp-min
                                                          inet:dscp
             |     |     |        +--rw diffserv-action:dscp-max
                                                          inet:dscp
             |     |     |        +--rw diffserv-action:threshold
             |     |     |           +--rw (threshold-type)?
             |     |     |              +--:(size)
             |     |     |              |  +--rw diffserv-action:
                                                 threshold-size? uint64
             |     |     |              +--:(interval)
             |     |     |                 +--rw diffserv-action:
                                                 threshold-interval?
                                                                uint64
             |     |     +--:(random-detect)
             |     |        +--rw diffserv-action:random-detect-cfg
                                                     {aqm-wred-support}?
             |     |           +--rw diffserv-action:exp-weighting-const?
                                                                uint32
             |     |           +--rw diffserv-action:mode-aggregate?
                                                               boolean
             |     |           +--rw diffserv-action:ecn-enabled?
                                                               boolean
             |     |           +--rw diffserv-action:wred-profile*
                                                       [wred-profile-id]
             |     |              +--rw diffserv-action:wred-profile-id
                                                               uint16
             |     |              +--rw diffserv-action:wred-dscp*
                                                     [dscp-min dscp-max]
             |     |              |  +--rw diffserv-action:dscp-min
                                                          inet:dscp
             |     |              |  +--rw diffserv-action:dscp-max
                                                          inet:dscp
             |     |              +--rw diffserv-action:wred-min-thresh
             |     |              |  +--rw diffserv-action:threshold
             |     |              |     +--rw (threshold-type)?
             |     |              |        +--:(size)
             |     |              |        |  +--rw diffserv-action:
                                                     threshold-size?
                                                                uint64
             |     |              |        +--:(interval)
             |     |              |           +--rw diffserv-action:
                                                     threshold-interval?
                                                                uint64
             |     |              +--rw diffserv-action:wred-max-thresh
```

```
           |     |              |  +--rw diffserv-action:threshold
           |     |              |     +--rw (threshold-type)?
           |     |              |        +--:(size)
           |     |              |        |  +--rw diffserv-action:
                                                  threshold-size?
                                                              uint64
           |     |              |        +--:(interval)
           |     |              |           +--rw diffserv-action:
                                                  threshold-interval?
                                                              uint64
           |     |             +--rw diffserv-action:mark-probability?
                                                              uint32
           |     +--:(min-rate)
           |        +--rw diffserv-action:min-rate-cfg
           |           +--rw diffserv-action:min-rate?   uint64
           +--rw child-policy?                   leafref
                                    {hierarchial-policy-support}?
```

ietf-diffserv-action module contains set of diffserv actions which are augmented to ietf-diffserv-policy module and to ietf-diffserv-target module. Marking sets Diffserv codepoint value in the classified packet. Color-aware and Color-blind meters can be configured. Action counters are defined as grouping and are currently not augmented to any diffserv module.

ietf-diffserv-target module contains reference of diffserv-policy for either direction of network traffic and is augmented to ietf-interfaces [RFC7223] module.

```
module: ietf-diffserv-target
augment /if:interfaces/if:interface:
+--rw diffserv-target-entry* [direction policy-name]
   +--rw policy-name                        string
   +--rw policy-descr?                      string
   +--rw direction                          identityref
   +--rw diffserv-target-classifier-entry*
             [classifier-entry-name parent-path]
                 {target-inline-policy-config}?
      +--rw classifier-entry-name           leafref
      +--rw parent-path                     string
      +--rw classifier-entry-inline?        boolean
      +--rw classifier-entry-filter-oper?   identityref
      +--rw filter-entry* [filter-type filter-logical-not]
                                {policy-inline-classifier-config}?
      |  +--rw filter-type                   identityref
      |  +--rw filter-logical-not            boolean
      |  +--rw (filter-param)?
```

```
          |     +--:(dscp)
          |     |  +--rw dscp-cfg* [dscp-min dscp-max]
          |     |     +--rw dscp-min    inet:dscp
          |     |     +--rw dscp-max    inet:dscp
          |     +--:(source-ip-address)
          |     |  +--rw source-ip-address-cfg* [source-ip-addr]
          |     |     +--rw source-ip-addr    inet:ip-prefix
          |     +--:(destination-ip-address)
          |     |  +--rw destination-ip-address-cfg*
                                  [destination-ip-addr]
          |     |     +--rw destination-ip-addr    inet:ip-prefix
          |     +--:(source-port)
          |     |  +--rw source-port-cfg*
                                  [source-port-min source-port-max]
          |     |     +--rw source-port-min    inet:port-number
          |     |     +--rw source-port-max    inet:port-number
          |     +--:(destination-port)
          |     |  +--rw destination-port-cfg*
                            [destination-port-min destination-port-max]
          |     |     +--rw destination-port-min    inet:port-number
          |     |     +--rw destination-port-max    inet:port-number
          |     +--:(protocol)
          |     |  +--rw protocol-cfg* [protocol-min protocol-max]
          |     |     +--rw protocol-min    uint8
          |     |     +--rw protocol-max    uint8
          |     +--:(flow-label)
          |        +--rw flow-label-cfg* [flow-label-min flow-label-max]
          |           +--rw flow-label-min    uint32
          |           +--rw flow-label-max    uint32
          +--rw classifier-action-entry-cfg* [action-type]
             +--rw action-type                          identityref
             +--rw (action-cfg-params)?
                +--:(marking)
                |  +--rw diffserv-action:marking-cfg
                |     +--rw diffserv-action:dscp?   inet:dscp
                +--:(priority)
                |  +--rw diffserv-action:priority-cfg
                |     +--rw diffserv-action:priority-level?   uint8
                |     +--rw diffserv-action:priority-rate?    uint64
                +--:(meter)
                |  +--rw diffserv-action:meter-cfg
                |     +--rw diffserv-action:meter-list* [meter-id]
                |        +--rw diffserv-action:meter-id          uint16
                |        +--rw diffserv-action:meter-rate?       uint64
                |        +--rw (burst-type)?
                |        |  +--:(size)
                |        |  |  +--rw diffserv-action:burst-size? uint64
                |        |  +--:(interval)
```

```
                |        |     +--rw diffserv-action:burst-interval?
                                                          uint64
                |        +--rw diffserv-action:color
                |        |  +--rw diffserv-action:classifier-entry-name?
                                                          string
                |        |  +--rw diffserv-action:classifier-entry-descr?
                                                         string
                |        |  +--rw diffserv-action:
                                  classifier-entry-filter-operation?
                                                       identityref
                |        +--rw diffserv-action:meter-action-type?
                                                       identityref
                |        +--rw (val)?
                |           +--:(meter-action-mark)
                |           |  +--rw diffserv-action:dscp?
                                            inet:dscp
                |           +--:(meter-action-drop)
                |              +--rw diffserv-action:drop-action?
                                            boolean
                +--:(max-rate)
                |  +--rw diffserv-action:max-rate-cfg
                |     +--rw diffserv-action:absolute-rate?    uint64
                |     +--rw (burst-type)?
                |        +--:(size)
                |        |  +--rw diffserv-action:burst-size? uint64
                |        +--:(interval)
                |           +--rw diffserv-action:burst-interval? uint64
                +--:(algorithmic-drop)
                |  +--rw (drop-algorithm)?
                |     +--:(always-drop)
                |     |  +--rw diffserv-action:drop-cfg
                |     |     +--rw diffserv-action:drop-action?   boolean
                |     +--:(tail-drop)
                |     |  +--rw diffserv-action:tail-drop-cfg
                |     |     +--rw diffserv-action:qlimit-dscp-thresh*
                                                  [dscp-min dscp-max]
                |     |        +--rw diffserv-action:dscp-min   inet:dscp
                |     |        +--rw diffserv-action:dscp-max   inet:dscp
                |     |        +--rw diffserv-action:threshold
                |     |           +--rw (threshold-type)?
                |     |              +--:(size)
                |     |              |  +--rw diffserv-action:
                                              threshold-size?   uint64
                |     |              +--:(interval)
                |     |                 +--rw diffserv-action:
                                              threshold-interval? uint64
                |     +--:(random-detect)
                |        +--rw diffserv-action:random-detect-cfg
```

```
                                                  {aqm-wred-support}?
               |          +--rw diffserv-action:exp-weighting-const?
                                                                uint32
               |          +--rw diffserv-action:mode-aggregate?
                                                              boolean
               |          +--rw diffserv-action:ecn-enabled?
                                                              boolean
               |          +--rw diffserv-action:wred-profile*
                                                   [wred-profile-id]
               |             +--rw diffserv-action:wred-profile-id
                                                                uint16
               |             +--rw diffserv-action:wred-dscp*
                                                  [dscp-min dscp-max]
               |             |  +--rw diffserv-action:dscp-min
                                                            inet:dscp
               |             |  +--rw diffserv-action:dscp-max
                                                            inet:dscp
               |             +--rw diffserv-action:wred-min-thresh
               |             |  +--rw diffserv-action:threshold
               |             |     +--rw (threshold-type)?
               |             |        +--:(size)
               |             |        |  +--rw diffserv-action:
                                              threshold-size? uint64
               |             |        +--:(interval)
               |             |           +--rw diffserv-action:
                                              threshold-interval?
                                                                uint64
               |             +--rw diffserv-action:wred-max-thresh
               |             |  +--rw diffserv-action:threshold
               |             |     +--rw (threshold-type)?
               |             |        +--:(size)
               |             |        |  +--rw diffserv-action:
                                              threshold-size? uint64
               |             |        +--:(interval)
               |             |           +--rw diffserv-action:
                                              threshold-interval?
                                                                uint64
               |             +--rw diffserv-action:mark-probability?
                                                                uint32
               +--:(min-rate)
                  +--rw diffserv-action:min-rate-cfg
                     +--rw diffserv-action:min-rate?   uint64
```

## 5. Diffserv Modules

### 5.1. IETF-DIFFSERV-CLASSIFIER

```
module ietf-diffserv-classifier {
  yang-version 1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-diffserv-classifier";
  prefix diffserv-classifier;

  import ietf-inet-types {
    prefix inet;
  }

  revision 2014-09-07 {
    description
      "First revision of diffserv based classifier";
  }

  feature policy-inline-classifier-config {
    description
      " This feature allows classifier configuration directly
        under policy.";
  }

  identity filter-type {
    description
      " This is identity of base filter-type";
  }

  identity dscp {
    base filter-type;
  }

  identity source-ip-address {
    base filter-type;
  }

  identity destination-ip-address {
    base filter-type;
  }

  identity source-port {
    base filter-type;
  }

  identity destination-port {
    base filter-type;
  }
```

```
     identity protocol {
       base filter-type;
     }

     identity flow-label {
       base filter-type;
     }

     identity classifier-entry-filter-operation-type {
       description
         "Classifier entry filter logical operation";
     }

     identity match-any-filter {
       base classifier-entry-filter-operation-type;
       description
         "Classifier entry filter logical OR operation";
     }

     identity match-all-filter {
       base classifier-entry-filter-operation-type;
       description
         "Classifier entry filter logical AND operation";
     }

     grouping filters {
       leaf filter-type {
         type identityref {
           base filter-type;
         }
         description
           "This leaf defines type of the filter";
       }
       leaf filter-logical-not {
         type boolean;
         description
           "
            This is logical-not operator for a filter. When true, it
            indicates filter looks for absence of a pattern defined
            by the filter
           ";
       }
       choice filter-param {
         case dscp {
           list dscp-cfg {
             key "dscp-min dscp-max";
             leaf dscp-min {
               type inet:dscp;
```

```
            }
            leaf dscp-max {
              type inet:dscp;
            }
          }
          description
            "Filter containing list of dscp ranges";
        }
        case source-ip-address {
          list source-ip-address-cfg {
            key "source-ip-addr";
            leaf source-ip-addr {
              type inet:ip-prefix;
            }
          }
          description
            "Filter containing list of source ip addresses";
        }
        case destination-ip-address {
          list destination-ip-address-cfg {
            key "destination-ip-addr";
            leaf destination-ip-addr {
              type inet:ip-prefix;
            }
          }
          description
            "Filter containing list of destination ip address";
        }
        case source-port {
          list source-port-cfg {
            key "source-port-min source-port-max";
            leaf source-port-min {
              type inet:port-number;
            }
            leaf source-port-max {
              type inet:port-number;
            }
          }
          description
            "Filter containing list of source-port ranges";
        }
        case destination-port {
          list destination-port-cfg {
            key "destination-port-min destination-port-max";
            leaf destination-port-min {
              type inet:port-number;
            }
            leaf destination-port-max {
```

```
              type inet:port-number;
            }
          }
          description
            "Filter containing list of destination-port ranges";
        }
        case protocol {
          list protocol-cfg {
            key "protocol-min protocol-max";
            leaf protocol-min {
              type uint8 {
                range "0..255";
              }
            }
            leaf protocol-max {
              type uint8 {
                range "0..255";
              }
            }
          }
          description
            "Filter Type Protocol";
        }
        case flow-label {
          list flow-label-cfg {
            key "flow-label-min flow-label-max";
            leaf flow-label-min {
              type uint32 {
                range "0..1048575";
              }
            }
            leaf flow-label-max {
              type uint32 {
                range "0..1048575";
              }
            }
          }
          description
            "Filter containing list of flow-label ranges";
        }
      }
    }

    grouping classifier-entry-generic-attr {
      leaf classifier-entry-name {
        type string;
        description
          "Diffserv classifier name";
```

```
        }
        leaf classifier-entry-descr {
          type string;
          description
            "Description of the class template";
        }
        leaf classifier-entry-filter-operation {
          type identityref {
            base classifier-entry-filter-operation-type;
          }
          default "match-any-filter";
        }
      }

      grouping classifier-entry-inline-attr {
        leaf classifier-entry-inline {
          type boolean;
          description
            "Indication of inline classifier entry";
          default "false";
        }
        leaf classifier-entry-filter-oper {
          type identityref {
            base classifier-entry-filter-operation-type;
          }
          default "match-any-filter";
        }
        list filter-entry {
          if-feature policy-inline-classifier-config;
          when "classifier-entry-inline == true";
          key "filter-type filter-logical-not";
          uses filters;
        }
      }

      container classifiers {
        description
          "list of classifier entry";
        list classifier-entry {
          key "classifier-entry-name";
          description
            "classifier entry template";
          uses classifier-entry-generic-attr;
          list filter-entry {
            key "filter-type filter-logical-not";
            uses filters;
          }
        }
```

```
    }
  }
```

## 5.2. IETF-DIFFSERV-POLICY

```
  module ietf-diffserv-policy {
    yang-version 1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-diffserv-policy";
    prefix diffserv-policy;

    import ietf-diffserv-classifier {
      prefix classifier;
    }

    revision 2014-09-07 {
      description
        "First revision of diffserv policy";
    }

    feature policy-template-support {
      description
        " This feature allows policy template to be configured";
    }

    feature hierarchial-policy-support {
      description
        " This feature allows hierarchial policy to be configured";
    }

    grouping policy-generic-attr {
      leaf policy-name {
        type string;
        description
          "Diffserv policy name";
      }
      leaf policy-descr {
        type string;
        description
          "Diffserv policy description";
      }
    }

    identity action-type {
      description
        "This base identity type defines action-types";
    }
```

```
     grouping classifier-action-entry-cfg {
       list classifier-action-entry-cfg {
         key "action-type";
         ordered-by user;
         leaf action-type {
           type identityref {
             base action-type;
           }
           description
             "This defines action type ";
         }
         choice action-cfg-params;
       }
     }

     container policies {
       description
         "list of policy templates";
       if-feature policy-template-support;
       list policy-entry {
         key "policy-name";
         description
           "policy template";
         uses policy-generic-attr;
         list classifier-entry {
           key "classifier-entry-name";
           ordered-by user;
           leaf classifier-entry-name {
             type string;
             description
               "Diffserv classifier entry name";
           }
           uses classifier:classifier-entry-inline-attr;
           uses classifier-action-entry-cfg;
           leaf child-policy {
             if-feature hierarchial-policy-support;
             type leafref {
               path "/policies/policy-entry/policy-name";
             }
           }
         }
       }
     }
   }
```

### 5.3. IETF-DIFFSERV-ACTION

```
module ietf-diffserv-action {
  namespace "urn:ietf:params:xml:ns:yang:ietf-diffserv-action";
  prefix diffserv-action;

  import ietf-interfaces {
    prefix if;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-diffserv-classifier {
    prefix diffserv-classifier;
  }
  import ietf-diffserv-policy {
    prefix diffserv-policy;
  }
  import ietf-diffserv-target {
    prefix diffserv-target;
  }

  revision 2014-09-07 {
    description
      "Initial revision for diffserv actions on network packets";
  }

  feature aqm-wred-support {
    description
      " This feature allows AQM WRED to be configured";
  }

  grouping dscp-range {
    leaf dscp-min {
      type inet:dscp;
    }
    leaf dscp-max {
      type inet:dscp;
    }
  }

  grouping burst {
    choice burst-type {
      case size {
        leaf burst-size {
          units "bytes";
          type uint64;
```

```
          }
        }
        case interval {
          leaf burst-interval {
            units "microsecond";
            type uint64;
          }
        }
      }
    }

    grouping threshold {
      container threshold {
        description
          "threshold";
        choice threshold-type {
          case size {
            leaf threshold-size {
              units "bytes";
              type uint64;
            }
          }
          case interval {
            leaf threshold-interval {
              units "microsecond";
              type uint64;
            }
          }
        }
      }
    }

    identity min-rate {
      base diffserv-policy:action-type;
    }

    identity marking {
      base diffserv-policy:action-type;
    }

    identity priority {
      base diffserv-policy:action-type;
    }

    identity meter {
      base diffserv-policy:action-type;
    }
```

```
     identity max-rate {
       base diffserv-policy:action-type;
     }

     identity algorithmic-drop {
       base diffserv-policy:action-type;
     }

     identity meter-action-type {
       description
         "conform/violate/exceed action type in a meter";
     }

     identity meter-action-drop {
       base meter-action-type;
     }

     identity meter-action-set {
       base meter-action-type;
     }

     grouping drop {
       leaf drop-action {
         type boolean;
       }
       description
         "the drop action";
     }

     grouping queuelimit {
       list qlimit-dscp-thresh {
         key "dscp-min dscp-max";
         uses dscp-range;
         uses threshold;
       }
     }

     grouping meter-action-params {
       leaf meter-action-type {
         type identityref {
           base meter-action-type;
         }
       }
       choice val {
         case meter-action-mark {
           uses marking;
           description
             "meter action: mark";
```

```
        }
        case meter-action-drop {
          description
            "meter action: drop";
          uses drop;
        }
      }
    }

    grouping meter {
      leaf meter-id {
        type uint16;
      }
      leaf meter-rate {
        units "bits-per-second";
        type uint64;
      }
      uses burst;
      container color {
        uses diffserv-classifier:classifier-entry-generic-attr;
      }
      uses meter-action-params;
    }

    grouping priority {
      leaf priority-level {
        type uint8;
        description
          "priority level";
      }
      leaf priority-rate {
        units "bits-per-second";
        type uint64;
      }
    }

    grouping min-rate {
      leaf min-rate {
        units "bits-per-second";
        type uint64;
      }
      description
        "min guanteed bandwidth";
    }

    grouping marking {
      leaf dscp {
        type inet:dscp;
```

```
      }
    }

    grouping max-rate {
      leaf absolute-rate {
        units "bits-per-second";
        type uint64;
      }
      uses burst;
    }

    grouping wred-threshold {
      container wred-min-thresh {
        uses threshold;
        description
          "Minimum threshold";
      }
      container wred-max-thresh {
        uses threshold;
        description
          "Maximum threshold";
      }
      leaf mark-probability {
        type uint32 {
          range "1..1000";
        }
        description
          "Mark probability";
      }
    }

    grouping randomdetect {
      leaf exp-weighting-const {
        type uint32;
        description
          "Exponential weighting constant factor for wred profile ";
      }
      leaf mode-aggregate {
        type boolean;
        default "false";
        description
          "
           Indicates aggregate mode or non-aggregate mode.
           Non-aggregate mode by default creates sub-class for each
           code-point value with different min and max threshold.
           Aggregate mode defaults to only one subclass unless
           explicitly configured by the user
          ";
```

```
        }
        leaf ecn-enabled {
          type boolean;
          default "false";
        }
        list wred-profile {
          key "wred-profile-id";
          leaf wred-profile-id {
            type uint16;
          }
          list wred-dscp {
            key "dscp-min dscp-max";
            uses dscp-range;
          }
          uses wred-threshold;
        }
      }

      augment "/diffserv-policy:policies/diffserv-policy:policy-entry
               /diffserv-policy:classifier-entry
               /diffserv-policy:classifier-action-entry-cfg
               /diffserv-policy:action-cfg-params" {
        case marking {
          container marking-cfg {
            uses marking;
          }
        }
        case priority {
          container priority-cfg {
            uses priority;
          }
        }
        case meter {
          container meter-cfg {
            list meter-list {
              key "meter-id";
              uses meter;
            }
          }
        }
        case max-rate {
          container max-rate-cfg {
            uses max-rate;
          }
        }
        case algorithmic-drop {
          choice drop-algorithm {
            case always-drop {
```

```
            container drop-cfg {
              uses drop;
            }
          }
          case tail-drop {
            container tail-drop-cfg {
              uses queuelimit;
            }
          }
          case random-detect {
            container random-detect-cfg {
              if-feature aqm-wred-support;
              uses randomdetect;
            }
          }
        }
      }
      case min-rate {
        container min-rate-cfg {
          uses min-rate;
        }
      }
    }
    augment "/if:interfaces/if:interface
             /diffserv-target:diffserv-target-entry
             /diffserv-target:diffserv-target-classifier-entry
             /diffserv-target:classifier-action-entry-cfg
             /diffserv-target:action-cfg-params" {
      case marking {
        container marking-cfg {
          uses marking;
        }
      }
      case priority {
        container priority-cfg {
          uses priority;
        }
      }
      case meter {
        container meter-cfg {
          list meter-list {
            key "meter-id";
            uses meter;
          }
        }
      }
      case max-rate {
        container max-rate-cfg {
```

```
            uses max-rate;
          }
        }
        case algorithmic-drop {
          choice drop-algorithm {
            case always-drop {
              container drop-cfg {
                uses drop;
              }
            }
            case tail-drop {
              container tail-drop-cfg {
                uses queuelimit;
              }
            }
            case random-detect {
              container random-detect-cfg {
                if-feature aqm-wred-support;
                uses randomdetect;
              }
            }
          }
        }
        case min-rate {
          container min-rate-cfg {
            uses min-rate;
          }
        }
      }
    }
```

## 5.4. IETF-DIFFSERV-TARGET

```
   module ietf-diffserv-target {
     yang-version 1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-diffserv-target";
     prefix diffserv-target;

     import ietf-interfaces {
       prefix if;
     }
     import ietf-diffserv-classifier {
       prefix classifier;
     }
     import ietf-diffserv-policy {
       prefix policy;
     }
```

```
     revision 2014-09-07 {
       description
         "First revision diffserv based policy applied to a target";
     }

     identity direction {
       description
         "This is identity of traffic direction";
     }

     identity inbound {
       base direction;
       description
         "Direction of traffic coming into the network entry";
     }

     identity outbound {
       base direction;
       description
         "Direction of traffic going out of the network entry";
     }

     feature target-inline-policy-config {
       description
         "This feature allows the policy configuration directly
          under a target.";
     }

     grouping policy-target-generic-attr {
       uses policy:policy-generic-attr;
       leaf direction {
         type identityref {
           base direction;
         }
       }
     }

     grouping wred-class-stats {
       leaf early-drop-pkts {
         type uint64;
         description
           "Early drop packets ";
       }
       leaf early-drop-bytes {
         type uint64;
         description
           "Early drop bytes ";
       }
```

```
     }

     grouping classifier-entry-stats {
       container classifier-entry-statistics {
         config false;
         description
           "
            This container defines the classifier filter statistics
            pertaining to each classifier entry.
           ";
         leaf classified-pkts {
           type uint64;
           description
             " Number of total packets which filtered
               to the classifier-entry";
         }
         leaf classified-bytes {
           type uint64;
           description
             " Number of total bytes which filtered
               to the classifier-entry";
         }
         leaf classified-rate {
           units "bits-per-second";
           type uint64;
           description
             " Rate of average data flow through the
               classifier-entry";
         }
         container queuing-statistics {
           description
             "queue related statistics ";
           config false;
           leaf output-pkts {
             type uint64;
             description
               "Number of packets transmitted from queue ";
           }
           leaf output-bytes {
             type uint64;
             description
               "Number of bytes transmitted from queue ";
           }
           leaf queue-size-pkts {
             type uint64;
             description
               "Number of packets currently buffered ";
           }
```

```
          leaf queue-size-bytes {
            type uint64;
            description
              "Number of bytes currently buffered ";
          }
          leaf drop-pkts {
            type uint64;
            description
              "Total number of packets dropped ";
          }
          leaf drop-bytes {
            type uint64;
            description
              "Total number of bytes dropped ";
          }
          list wred-statistics {
            key "wred-profile-id";
            description
              "WRED statistics for a dscp range ";
            leaf wred-profile-id {
              type uint16;
            }
            uses wred-class-stats;
          }
        }
      }
    }

    grouping meter-action-stats {
      list meter-action-statistics {
        description
          "Meter statistics";
        config false;
        key "meter-id";
        leaf meter-id {
          type uint16;
        }
        leaf metered-pkts {
          type uint64;
          description
            "Number of packets counted by the meter";
        }
        leaf metered-bytes {
          type uint64;
          description
            "Bytes of packets counted by the meter";
        }
        leaf metered-rate {
```

```
          units "bits-per-second";
          type uint64;
          description
            "Traffic Rate measured by the meter";
        }
      }
    }

    augment "/if:interfaces/if:interface" {
      list diffserv-target-entry {
        key "direction policy-name";
        description
          "policy target for inbound or outbound direction";
        uses policy-target-generic-attr;
        list diffserv-target-classifier-entry {
          if-feature target-inline-policy-config;
          key "classifier-entry-name parent-path";
          ordered-by user;
          leaf classifier-entry-name {
            type leafref {
              path "/classifier:classifiers/classifier:classifier-entry
                    /classifier:classifier-entry-name";
            }
          }
          leaf parent-path {
            type string;
          }
          uses classifier:classifier-entry-inline-attr;
          uses policy:classifier-action-entry-cfg;
        }
      }
    }
  }
```

## 6. Security Considerations

## 7. Acknowledgement

The editor of this document wishes to thank Fred Baker for overviewing the document and provide useful comments, Andrew Mao for the guidance and support, Fred Yip and Aleksandr Zhdankin for helpful suggestions and contributions.

## 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.

[RFC2697] Heinanen, J. and R. Guerin, "A Single Rate Three Color Marker", RFC 2697, September 1999.

[RFC2698] Heinanen, J. and R. Guerin, "A Two Rate Three Color Marker", RFC 2698, September 1999.

[RFC2859] Fang, W., Seddigh, N., and B. Nandy, "A Time Sliding Window Three Colour Marker (TSWTCM)", RFC 2859, June 2000.

[RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, March 2002.

[RFC3260] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, April 2002.

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

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

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

## 8.2. Informative References

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.

## Appendix A. Open Items

The current model represents hierarchical QoS alike with the non-leaf and leaf nodes, in a scheduling hierarchy, without any restrictions of actions, such as AQM, that should not be allowed at non-leaf nodes. This is to be addressed in subsequent revisions.

## Authors' Addresses

Aseem Choudhary
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134
US

Email: asechoud@cisco.com

Shitanshu Shah
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134
US

Email: svshah@cisco.com

Mahesh Jethanandani
Ciena Corporation
3939 North 1st Street
San Jose, CA 95134
US

Email: mjethanandani@gmail.com

Gang Yan
Huawei Technologies
Huawei Bld., No. 156 Beiqing Rd
Beijing 100095
China

Email: yangang@huawei.com

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Rd
Beijing 100095
China

Email: Leo.liubing@huawei.com

Norm Strahle
Juniper Networks
1194 North Mathilda Avenue
Sunnyvale, CA 94089
US

Email: nstrahle@juniper.net

NETMOD
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

D. Bogdanovic
Juniper Networks
B. Claise
C. Moberg
Cisco Systmes, Inc.

March 09, 2015

# YANG model classification
# draft-bogdanovic-netmod-yang-model-classification-01

## Abstract

YANG became de facto standard language for data modeling in the industry. More and more groups uses YANG to create protocol and service models, both for configuration and operational models. Currently there is a lack of consistent terminology to categorize those models. A consistent terminology would help models categorization, assist in the analysis the YANG data modeling effort in the IETF and in the industry, and facilitate the YANG-related discussions between different groups.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

With more and more data models being described with YANG [RFC6020], a taxonomy for the models is needed. In this draft, authors try to propose a taxonomy for YANG models.

## 2. Problem Statement

YANG [RFC6020] became de-facto standard language for data modeling in the industry. Not only at the IETF, but also in multiple Standard Development Organizations, different consortia, ad hoc groups, and OSP. Therefore, many YANG models are being developed and published. Today, there is no classification of models, there are no clear guidelines on how to layer models on each other, or how to classify existing or new models. With this document, the authors are proposing a new way for YANG model classifications.

Acknowledging that the YANG became the de-facto standard language for data modeling, the Internet Engineering Steering Group (IESG) has been encouraging the working groups to use the NETCONF [RFC6241] and YANG standards for configuration, especially in new charters [Writable-MIB-Module-IESG-Statement].

YANG Models can classified according to two dimensions: based on the layer in the hierarchy of models, and based on the model type. Those two categories are covered in the next two sections.

## 3. First Dimension: Network YANG Data Model Layering

When developing models, there are two approaches possible, top down and bottom up. Top down approach is driven by business requirements and bottom up is driven by technological ones.

There are no hard requirements on how to create modeling, but it would be useful to have a classification and to create models that can be easily reused, as with this time and energy will be saved in future development. We should stimulate both development styles, bottom up and top down, as each has its benefits and groups to which a certain style will be more appealing than the other.

For layering purposes, we can classify data models into two layers:

Network Service YANG Model

Network Element YANG Model

Figure 1 displays example YANG models at different layers. By layering the models, it is easier to achieve reusability of existing lower layer models in higher level models and preventing duplication of same features modeled in different layers. When developing models per layers, it allows creating very focused groups in specific areas. As an example, creating protocol data definitions network equipment YANG data models should involve people that have intimate experience of implemention details. On the other hand, network service models are best developed by people experienced in network operations. Same network service, can be implemented and modeled using different protocol and feature YANG models.

```
                        +-----------------------+
                          |                       |
                          |        OSS/BSS        |
                          |                       |
                          +-----------------------+

      Network Service YANG data models ------------------------------

                  +------------+      +-------------+
                  |            |      |             |
                  |  - VPWS    |      |   - VPLS    |
                  |  - L2VPN   |      |   - L2VPN   |
                  |            |      |             |
                  +------------+      +-------------+


      Network Element YANG data models ---------------------------------

    +------------+    +------------+   +--------------+  +-----------+
    |            |    |            |   |              |  |           |
    |     MPLS   |    |    BGP     |   |  Interface   |  |  Routing  |
    |            |    |            |   |              |  |           |
    +------------+    +------------+   +--------------+  +-----------+
```

### 3.1. Network Service YANG Data Models

Network Service YANG models are created by network operators when they choose how to configure their network from technology point of view. They decide which technology is best match for their business needs and based on that create network service data models. With more and more configuration models being available, both vendor and standards one, network service model developer can create reusable components based on the configuration models, and use those service components data models to create end to end service model.

For example, http://datatracker.ietf.org/doc/draft-l3vpn-service-yang/ provides an abstracted view of the Layer 3 IP VPN service configuration components. It will be up to an orchestrator to take this as an input and use specific configurations models on the network elementlayer to configure the different network elements to deliver the service.

Network Service YANG models can be developed in multiple ways. Building them monolithic from vendor models or by combining one or more service components into an end to end service data model. It specifies complete service that is provided by the network operator.

Building monolithic network service model has an advantage of doing it fast, but at the expense of flexibility of updating the service later or equipment vendors. Such an end to end service can be VPLS/ VPWS L2VPN, IPsec, etc. If we take into example VPLS L2VPN service, it can be built as a single network service model or it can be built from several service components. VPWS L2VPN service can be built on top of MPLS or IP network core. When building such a network service model, network variations have to be taken into the account and by creating service components modle, such as MPLS, BGP service component models, it is easier to build a network service model, such as VPWS L2VPN.

### 3.2. Network Element YANG Data models

This is base model for all higher models. It fully describes protocol, such as OSPF [I-D.yeung-netmod-ospf], ISIS [I-D.ietf-isis-yang-isis-cfg] or feature, example access control list [I-D.ietf-netmod-acl-model]. The base model can be either vendor specific, which then describes vendor implementation of the protocol or feature or standard model. Key difference between those two is what is implemented. Vendor device model will always describe what is implemented, which can ber more or even less then in standard model. Standard model describes what is agreed in the industry to be accepted as base description.

## 4. Second Dimension: Model Type

At very high level, models can be divided into proprietary and standard. Each vendor, consortium, open source project can publish their models and those are considered proprietary models. When an SDO, such as IETF or ITU, publishes an accepted model document, then this is a standard model. There are use cases where a consortium has published work which de facto became standard, such as Linux kernel, but for the clarity in this document, authors are making a separation between models based on the above description.

Standard YANG Model:

Standard Extension YANG Model: .

Proprietary Extension to Standard YANG Model: As the Standard YANG Models contains a subset of all the Vendor Configuration Models, proprietary extensions must complement the Standard YANG Models to represent a Vendor Configuration Model.

Vendor Configuration Model: It describes all configurable capabilities of the device and what device vendor exposes for

configuration. The vendor configuration model can be CLI or YANG-based.

Proprietary YANG Model: A non Standard YANG Model.

As mentioned earlier in this document, there are two ways of designing models, top down and bottom up with one restriction. Everything is dependent on the vendor data model. That model describes all the possibilities and if model developers prefers, they can use vendor model only to design service components, network service and business service. Using vendor model provides all capabilities today, but it comes with restrictions of portability between vendors and to certain extent devices. On the other hand, only standard models and standard extensions can be used, but this might result in less feature rich or less efficient services. Service model developer has a choice to reuse service components or write a model completely based on vendor data model.

## 4.1. Standard YANG model

With YANG we have a common language, that enables different communities to express data models that are widely understandable without lot of additional explanation. This enables different groups, such as IETF, to standardize data models, defined as an IETF RFC, and vendors to support them, which will make it easier to for network operators to manage their network configuration programmatically. For example, A YANG Data Model for Interface Management [RFC7223], or the Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols [RFC6728].

## 4.2. Standard Extension YANG Model

Standard Extension is a the conditional portion of a Standard YANG Model, expressed with the feature, if-feature, augment YANG statements [RFC6020]. An example of such standard extension is policy based routing (PBR). PBR is found in many vendor implementations and have many common features, but not all vendors support PBR on all of their devices.

## 4.3. Proprietary Extension to Standard YANG Model

Proprietary extension is a conditional portion of a Standard YANG Model, expressed with feature, if-feature, augment YANG statements [RFC6020]. Proprietary extension can be a feature depending on harward platform capabilities and it is not available by other vendors. Such an example could be match condition for packet classification used for PBR.

### 4.4. Vendor configuration model

Base model for all other models is the vendor configuration model. It describes all configurable capabalities of the device and what device vendor exposes for configuration.

The standard configuration model is a subset of vendor configuration model. The standard configuration model can be broken into base model and standard extension models, where the base is common data model and standard extensions are standard features that are not implemented by all vendors. Example of standard base model is Access Control List and routing filter is a standard extension on ACL. Or another example: encryption algorithm is standard feature, but the different types, like md5, hmac-md5, hmac-sha1, etc are standard extensions, as it is not that all vendors have all encryption algorithm types implemented.

Although all vendors provide very similar functionality using standards, implementations are different. One of basic examples are dynamic routing protocols. We can see today two main types of routing protocol configuration.

protocol centric - all the protocol related config is contained with the protocol itself. Especially in case of multiple instances of the routing protocol running in different routing-instances (routing-instance as described in core routing model [I-D.ietf-netmod-routing-cfg]), all the routing-instance protocol config is contained in the default routing instance.

```
Router ospf 10
Default-metric 100
Address-family ipv4 vrf VRF1
    Network x.x.x.x area 0

Address-family ipv4 vrf VRF2
    Network x.x.x.x area 0

Address-family ipv4
    Network x.x.x.x area 1
```

In term of YANG model, the routing protocol configuration will be defined within the default routing-instance and the routing-protocol config will contain multiple instances referring to other routing-instances.

VRF centric - All the protocol related config for a routing-instance is contained within this routing-instance.

```
Routing-instance VRF1 {
  Protocols isis {

  }
}
Routing-instance VRF2 {
  Protocols isis {

  }
}
```

In term of YANG model, the routing protocol configuration for a routing-instance will be defined within the associated routing-instance.

The bottom line message is that, even if YANG models are standardized, they will provide different CLI outcomes, simply because the CLI among vendors is not standardized.

### 4.5. Proprietary YANG Model

While waiting for the Standard YANG Models to be published, the different vendors might offer Proprietary YANG Models.

## 5. Typical Architecture

```
          +-------------------------------------------------------+
          |                      OSS/BSS                          |
          +-------------------------------------------------------+

          +-------------------------------------------------------+
          | Orchestrator                                          |
          |   +-----------------------------------------------+   |
          |   | network service model                         |   |
          |   | +-------------------+   +-------------------+ |   |
          |   | | Service Component |   | Service Component | |   |
          |   | | Model             |   | Model             | |   |
          |   | |                   |   |                   | |   |
          |   | +-------------------+   +-------------------+ |   |
          |   +-----------------------------------------------+   |
              +-----------------------------------------------------+

              +-----------------------------------------------------+
          | Network element                                       |
          |                             |                         |
          |  +----------------------+  |  +------------------+   |
          |  | Standard YANG model  |  |  |   Proprietary    |   |
          |  |                      |  |  |   YANG model     |   |
              |  +----------------------+  |  |                  |   |
          |                             |  |                  |   |
          |  +----------------------+  |  |                  |   |
          |  | Proprietary Extension |  |  |                  |   |
          |  | To YANG Standard     |  |  |                  |   |
          |  | Model                |  |  |                  |   |
          |  +----------------------+  |  +------------------+   |
          |                             |                         |
          |  +-----------------------------------------------+   |
          |  |          Vendor Configuration Model           |   |
          |  +-----------------------------------------------+   |
          +-------------------------------------------------------+
```

## 6. IETF, Other SDOs, and open source

IETF, as a standard defining organization (SDO), is well positioned to standardize Network Element YANG models. With a wide range of expertise found within its WGs focused on those technology definitions. As IETF participants implement those protocols, they have deep expertise about the implementation and finding a common base standard configuration model between vendors should be a very viable goal.

In some situation where the protocols are standardized by different SDOs, those SDOs should be responsible for its YANG data modeling

effort. For example, the IETF has transferred the responsibility for some IEEE technology-related MIB modules to the IEEE 802.1 and 802.3 Working Group [RFC4663], [RFC7448]. Similarly, the IEEE should be responsible for similar YANG data modeling efforts.

Although many network operators participate in IETF work, developing higher level models requires network operations expertise. If such teams within right WG can be formed, then some service models can be developed within IETF, but some groups, like Metro Ethernet Forum or CableLabs could be better positioned for service modeling.

Today there are many open source projects and some of them are becoming de facto standards, like the Linux kernel. Many such open source projecs, like Open Daylight, OpenStack, etc, are doing very good work and their work is being accepted and deployed in production environments. They bring a lot of very valuable experience to other groups. From IETF perspective, if there is such a work present, it can be used as a very good starting point for modeling within IETF.

## 7. Security Considerations

At this stage, authors of the draft didn’t look into security considerations.

## 8. IANA Considerations

This document requests no action by IANA.

## 9. Acknowledgements

Thanks to David Ball for his enlightenments on Metro Ethernet Forum service aspects.

## 10. Change log [RFC Editor: Please remove]

version 1: restructure the document, add the two dimensions, add the interaction with the different SDOs and opensource projects, add the definitions.

## 11. References

### 11.1. Normative References

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

## 11.2. Informative References

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, J., and L. Lhotka, "YANG Data Model for ISIS protocol", draft-ietf-isis-yang-isis-cfg-01 (work in progress), October 2014.

[I-D.ietf-netmod-acl-model]
Bogdanovic, D., Sreenivasa, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-01 (work in progress), February 2015.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L., "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-16 (work in progress), October 2014.

[I-D.yeung-netmod-ospf]
Yeung, D., Qu, Y., Zhang, J., Bogdanovic, D., and K. Sreenivasa, "Yang Data Model for OSPF Protocol", draft-yeung-netmod-ospf-02 (work in progress), October 2014.

[RFC4663] Harrington, D., "Transferring MIB Work from IETF Bridge MIB WG to IEEE 802.1 WG", RFC 4663, September 2006.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC6728] Muenz, G., Claise, B., and P. Aitken, "Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols", RFC 6728, October 2012.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7448] Taylor, T. and D. Romascanu, "MIB Transfer from the IETF to the IEEE 802.3 WG", RFC 7448, February 2015.

[Writable-MIB-Module-IESG-Statement]
"Writable MIB Module IESG Statement", <https://www.ietf
.org/iesg/statement/writable-mib-module.html>.

## Authors' Addresses

Dean Bogdanovic
Juniper Networks

Email: deanb@juniper.net

Benoit Claise
Cisco Systmes, Inc.

Email: bclaise@cisco.com

Carl Moberg
Cisco Systmes, Inc.

Email: camoberg@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 4, 2015

J. Dong
M. Chen
Huawei Technologies
March 3, 2015

# A Data Model for Network Inventories
# draft-dong-i2rs-network-inventory-00

## Abstract

This document defines a YANG data model for network inventories.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 4, 2015.

## Copyright Notice

Table of Contents



## 1. Introduction

This document defines the Yang [RFC6020] data model for network inventory information. It augments the abstract (base) network Yang model defined in [I-D.clemm-i2rs-yang-network-topo] with inventory information of the network entities. This inventory model also describes the protocol independent features, functions and capabilities of the network entities and their components. This network inventory data model may be augmented to describe detailed inventory information and characteristics.

This network inventory data model can be used by applications in several ways, such as:

- o to obtain a complete view of the network inventory information;

- o to enable/disable some protocol independent functions with the characteristics specified in the data model;

- o to receive notifications of the changes in the network inventories.

The relationship between the inventory model and the abstract network model is shown in the diagram below:

```
                     +------------------------+
                     |                        |
                     | Abstract Network Model |
                     |                        |
                     +------------------------+
                                 |
                          +-------+-------+
                          |               |
                          V               V
                   +------------+  +------------+
                   |  Abstract  |  |  Network   |
                   |  Topology  |  | Inventory  |
                   |   Model    |  |   Model    |
                   +------------+  +------------+
```

Figure 1. Relationship of network inventory model and network model

## 2. Network Inventory Model Structure

Network inventory refers to the network entities used in building the network. Normally each network entity consists of a series of cards and interfaces, both the network entity and its components have a set of features and support a set of functions. Those protocol independent features and functions are described in this inventory model. For the interface part, some contents of the ietf-interfaces module as defined in [RFC7223] are reused.

The structure of "network-inventory" data model is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses enclose choice and case nodes.

```
module: network-inventory
augment /ntw:network/ntw:node:
   +--rw node-name?              string
   +--rw node-description?       string
   +--rw hardware-version?       string
   +--rw software-version?       string
   +--ro capability-supported*   node-capability
   +--rw capability-enabled*     node-capability
   +--rw node-status?            node-status
   +--rw power-consumption?      uint32
   +--rw card* [card-id]
   |  +--rw card-id             inet:uri
   |  +--rw card-description?   string
   |  +--rw card-type?          identityref
   |  +--rw card-capability*    card-capability
   |  +--rw admin-status?       card-status
   +--rw interface* [if-name]
      +--rw if-name         if:interface-ref
      +--rw description?    string
      +--rw card-ref?       leafref
      +--rw if-type?        identityref
      +--rw phys-address?   yang:phys-address
      +--rw transceiver?    identityref
      +--rw throughput?     uint64
      +--rw admin-status?   enumeration
      +--ro oper-status?    enumeration
```

## 3. Network Inventory Yang Module

```
module network-inventory {
    yang-version 1;
    namespace "urn:TBD:params:xml:ns:yang:network-inventory";
    // replace with IANA namespace when assigned

    prefix "netinv";

    import ietf-yang-types {
      prefix yang;
    }

    import ietf-inet-types {
      prefix inet;
    }

    import ietf-interfaces {
      prefix if;
    }
```

```
    import iana-if-type {
      prefix ift;
    }

    import network {
      prefix ntw;
    }

    organization "TBD";
    contact "I-D editor: jie.dong@huawei.com";
    description
    "This module defines a model for the network inventory.
     Key design considerations are as follows:
     A network consists of a set of network nodes.
     A network node can support a set of functions & capabilities.
     A network node consists of a set of cards and interfaces.
     A network node has a set of parameters to reflect its status.";

    revision "2015-02-10" {
        description "Initial revision";
        reference "TBD";
    }

   /*
    * Typedefs
    */

    typedef node-capability {
        type enumeration {
            enum ip-unicast {
                value 1;
            }
            enum ip-multicast {
                value 2;
            }
            enum mpls {
                value 3;
            }
            enum mpls-traffic-engineering {
                value 4;
            }
            enum ethernet-bridging {
                value 5;
            }
            enum cgn {
                value 6;
            }
```

```
        }
    }

    typedef node-status {
        type enumeration {
            enum other {
                value 0;
            }
            enum operating {
                value 1;
            }
            enum idle {
                value 2;
            }
            enum dormant {
                value 3;
            }
        }
    }

    typedef card-capability {
        type enumeration {
            enum ip-forwarding {
                value 1;
            }
            enum ethernet-bridging {
                value 2;
            }
            enum mpls-forwarding {
                value 3;
            }
            enum traffic-accounting {
                value 4;
            }
            enum caching {
                value 5;
            }
            enum cgn {
                value 6;
            }
        }
    }

    typedef card-status {
        type enumeration {
            enum operating {
                value 1;
            }
```

```
            enum idle {
                value 2;
            }
            enum dormant {
                value 3;
            }
        }
    }


   /*
    * Features
    */


   /*
    * Identities
    */

    identity card-type {
        description
          "Base identity from which specific card types are
           derived.";
    }

    identity control-card {
        base card-type;
        description
          "control-card";
    }

    identity line-card {
        base card-type;
        description
          "line-card";
    }

    identity fabric-card {
        base card-type;
        description
          "fabric-card";
    }

    identity transceiver-type {
        description
          "Base identity from which specific
           transceiver types are derived.";
    }
```

```
   /*
    * Data nodes
    */

    augment "/ntw:network/ntw:node" {

       leaf node-name {
           type string;
           description
             "The name of the node";
       }

       leaf node-description {
           type string;
       }

       leaf hardware-version {
           type string;
       }

       leaf software-version {
           type string;
       }

       leaf-list capability-supported {
           config false;
           type node-capability;
           description
            "A list of capabilities the node can support.";
       }

       leaf-list capability-enabled {
           type node-capability;
           description
             "A list of capabilities that has enabled.";
       }

       leaf node-status {
           type node-status;
       }

       leaf power-consumption {
           type uint32;
           units "watt";
       }

       list card {
           key "card-id";
```

```
            leaf card-id {
                type inet:uri;
            }

            leaf card-description {
                type string;
                description
                  "The description of the card";
            }

            leaf card-type {
                type identityref {
                    base card-type;
                }
            }

            leaf-list card-capability {
                type card-capability;
                description
                  "A list of capabilities the card supports.";
            }

            leaf admin-status {
                type card-status;
            }
        } //list-card

        list interface {
            key "if-name";

            leaf if-name {
                type if:interface-ref;
                description
                  "A reference to an interface";
            }

            leaf description {
                type string;
            }

            leaf card-ref {
                type leafref {
                    path "/ntw:network/ntw:node/card/netinv:card-id";
                }
                description
                  "A reference to the card which the interface belongs to.";
            }
```

```
            leaf if-type {
                type identityref {
                    base ift:iana-interface-type;
                }
            }

            leaf phys-address {
                type yang:phys-address;
                description
                  "The inteface’s physical address.";
            }

            leaf transceiver {
                type identityref {
                    base transceiver-type;
                }
                description
                  "The interface’s transceiver type";
            }

            leaf throughput {
                description
                  "The maximum throughput of the interface.";
                type uint64;
                units "bits/second";
            }

            leaf admin-status {
                type enumeration {
                    enum up {
                      value 1;
                      description
                        "Ready to pass packets.";
                    }
                    enum down {
                      value 2;
                      description
                        "Not ready to pass packets and not in test mode.";
                    }
                    enum testing {
                        value 3;
                        description
                          "In some test mode.";
                    }
                }
            }

            leaf oper-status {
```

```
                config false;
                type enumeration {
                    enum up {
                      value 1;
                      description
                        "Ready to pass packets.";
                    }
                    enum down {
                      value 2;
                      description
                        "The interface does not pass any packets.";
                    }
                    enum testing {
                      value 3;
                      description
                        "In some test mode.  No operational packets can
                         be passed.";
                    }
                    enum unknown {
                      value 4;
                      description
                        "Status cannot be determined for some reason.";
                    }
                    enum dormant {
                      value 5;
                      description
                        "Waiting for some external event.";
                    }
                    enum not-present {
                      value 6;
                      description
                        "Some component (typically hardware) is missing.";
                    }
                    enum lower-layer-down {
                      value 7;
                      description
                        "Down due to state of lower-layer interface(s).";
                    }
                }
            }

        }//list interface
    }


   /*
    * Notifications: to be added
    */
```

```
} //module
```

## 4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 5. Security Considerations

The transport protocol used for sending information of this data model MUST support authentication and SHOULD support encryption. The data-model by itself does not create any security implications.

## 6. Acknowledgements

TBD

## 7. Normative References

[I-D.clemm-i2rs-yang-network-topo]
Clemm, A., Medved, J., Varga, R., Tkacik, T., Bahadur, N., and H. Ananthakrishnan, "A Data Model for Network Topologies", draft-clemm-i2rs-yang-network-topo-02 (work in progress), December 2014.

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

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

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

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

## Authors' Addresses

Jie Dong
Huawei Technologies
Huawei Campus, No. 156 Beiqing Rd.
Beijing 100095
China

Email: jie.dong@huawei.com

Mach(Guoyi) Chen
Huawei Technologies
Huawei Campus, No. 156 Beiqing Rd.
Beijing 100095
China

Email: mach.chen@huawei.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2015

D. Bogdanovic
Juniper Networks
K. Sreenivasa
Brocade Communications System
L. Huang
D. Blair
Cisco Systems
March 5, 2015

# Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-02

## Abstract

This document describes a data model of Access Control List (ACL) basic building blocks.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2015.

Table of Contents



## 1. Introduction

Access Control List (ACL) is one of the basic elements to configure device forwarding behavior. It is used in many networking concepts such as Policy Based Routing, Firewalls etc.

An ACL is an ordered set of rules that is used to filter traffic on a networking device. Each rule is represented by an Access Control Entry (ACE).

Each ACE has a group of match criteria and a group of action criteria.

The match criteria consist of a tuple of packet header match criteria and metadata match criteria.

- o Packet header matches apply to fields visible in the packet such as address or class of service or port numbers.

- o Metadata matches apply to fields associated with the packet but not in the packet header such as input interface or overall packet length

The actions specify what to do with the packet when the matching criteria is met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding.The list of potential actions is endless depending on the innovations of the networked devices.

## 1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

AFI: Address Field Identifier

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

TCP: Transmission Control Protocol

# 2. Problem Statement

This document defines a YANG [RFC6020] data model for the configuration of ACLs. It is very important that model can be easily reused between vendors and between applications.

ACL implementations in every device may vary greatly in terms of the filter constructs and actions that they support. Therefore this draft proposes a simple model that can be augmented by vendor proprietary models.

# 3. Design of the ACL Model

Although different vendors have different ACL data models, there is a common understanding of what access control list (ACL) is. A network system usually have a list of ACLs, and each ACL contains an ordered

list of rules, also known as access list entries - ACEs. Each ACE has a group of match criteria and a group of action criteria. The match criteria consist of packet header matching and metadata matching. Packet header matching applies to fields visible in the packet such as address or class of service or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs. The model also includes overall operational state for the ACL and operational state for each ACE, targets where the ACL applied. One ACL can be applied to multiple targets within the device, such as interfaces of a networked device, applications or features running in the device, etc. When applied to interfaces of a networked device, the ACL is applied in a direction which indicates if it should be applied to packet entering (input) or leaving the device (output).

This draft tries to address the commonalities between all vendors and create a common model, which can be augmented with proprietary models. The base model is very simple and with this design we hope to achieve needed flexibility for each vendor to extend the base model.

### 3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-acl", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-acl" uses groupings in "ietf-packet-fields". The "ietf-packet-fields" modules can easily be extended to reuse definitions from other modules such as IPFIX [RFC5101] or migrate proprietary augmented module definitions into the standard module.

```
module: ietf-acl
+--rw access-lists
+--rw access-list* [access-control-list-name]
+--rw access-control-list-name         string
+--rw access-control-list-type?        access-control-list-type
+--ro access-control-list-oper-data
|  +--ro (targets)?
|     +--:(interface-name)
|        +--ro interface-name*   string
+--rw access-list-entries
+--rw access-list-entry* [rule-name]
```

```
+--rw rule-name                        string
+--rw matches
|  +--rw (access-list-entries-type)?
|  |  +--:(access-list-entries-ip)
|  |  |  +--rw source-port-range
|  |  |  |  +--rw lower-port    inet:port-number
|  |  |  |  +--rw upper-port?   inet:port-number
|  |  |  +--rw destination-port-range
|  |  |  |  +--rw lower-port    inet:port-number
|  |  |  |  +--rw upper-port?   inet:port-number
|  |  |  +--rw dscp?                           inet:dscp
|  |  |  +--rw protocol?                       uint8
|  |  |  +--rw (access-list-entries-ip-version)?
|  |  |     +--:(access-list-entries-ipv4)
|  |  |     |  +--rw destination-ipv4-network?       inet:ipv4-prefix
|  |  |     |  +--rw source-ipv4-network?            inet:ipv4-prefix
|  |  |     +--:(access-list-entries-ipv6)
|  |  |        +--rw destination-ipv6-network?       inet:ipv6-prefix
|  |  |        +--rw source-ipv6-network?            inet:ipv6-prefix
|  |  |        +--rw flow-label?                     inet:ipv6-flow-label
|  |  +--:(access-list-entries-eth)
|  |     +--rw destination-mac-address?        yang:mac-address
|  |     +--rw destination-mac-address-mask?   yang:mac-address
|  |     +--rw source-mac-address?             yang:mac-address
|  |     +--rw source-mac-address-mask?        yang:mac-address
|  +--rw input-interface?                string
|  +--rw absolute
|     +--rw start?    yang:date-and-time
|     +--rw end?      yang:date-and-time
|     +--rw active?   boolean
+--rw actions
|  +--rw (packet-handling)?
|     +--:(deny)
|     |  +--rw deny?     empty
|     +--:(permit)
|        +--rw permit?   empty
+--ro access-list-entries-oper-data
            +--ro match-counter?   yang:counter64
```

## 4. ACL YANG Models

### 4.1. IETF-ACL module

"ietf-acl" is the standard top level module for Access lists. It has a container for "access-list" to store access list information. This container has information identifying the access list by a name("acl-name") and a list("access-list-entries") of rules associated with the "acl-name". Each of the entries in the list("access-list-entries")

indexed by the string "rule-name" have containers defining "matches" and "actions". The "matches" define criteria used to identify patterns in "ietf-packet-fields". The "actions" define behavior to undertake once a "match" has been identified.

```
<CODE BEGINS>file "ietf-acl@2015-03-04.yang"
module ietf-acl {
  yang-version 1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-acl";

  prefix access-control-list;


  import ietf-yang-types {
    prefix "yang";

  }

  import ietf-packet-fields {
    prefix "packet-fields";
  }

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

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

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

    WG Chair: Tom Nadeau
    tnadeau@lucidvision.com

    Editor: Dean Bogdanovic
    deanb@juniper.net

    Editor: Kiran Agrahara Sreenivasa
    kkoushik@brocade.com

    Editor: Lisa Huang
    yihuan@cisco.com

    Editor: Dana Blair
    dblair@cisco.com";
```

```
  description
    "This YANG module defines a component that describing the
    configuration of Access Control Lists (ACLs).

    Copyright (c) 2015 IETF Trust and the persons identified as
    the document authors.  All rights reserved.

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


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

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

  revision 2015-03-04 {
    description "Base model for Network Access Control List (ACL).";
    reference
      "RFC XXXX: Network Access Control List (ACL)
      YANG Data  Model";
  }

  identity access-control-list-base {
    description "Base access control list type for all access control list type
      identifiers.";
  }

  identity IP-access-control-list {
    base "access-control-list:access-control-list-base";
    description "IP-access control list is common name for layer 3 and 4 access
    control list types. It is common among vendors to call 3-tupple or 5 tupple
    IP access control lists";
  }

  identity eth-access-control-list {
    base "access-control-list:access-control-list-base";
    description "Ethernet access control list is name for layer 2 Ethernet
    technology access control list types, like 10/100/1000baseT or WiFi
    access control list";
  }

  typedef access-control-list-type {
```

```
    type identityref {
      base "access-control-list-base";
    }
    description
      "This type is used to refer to an Access Control List
      (ACL) type";
  }

  typedef access-control-list-ref {
    type leafref {
      path "/access-lists/access-list/access-control-list-name";
    }
    description "This type is used by data models that need to referenced an
      access control list";

  }

  container access-lists {
    description
      "This is top level container for Access Control Lists. It can have one
        or more Access Control List.";

    list access-list {
      key access-control-list-name;
      description "An access list (acl) is an ordered list of
        access list entries (ACE). Each access control entries has a
        list of match criteria, and a list of actions.
        Since there are several kinds of access control lists
        implemented with different attributes for
        each and different for each vendor, this
        model accommodates customizing access control lists for
        each kind and for each vendor.";

      leaf access-control-list-name {
        type string;
        description "The name of access-list. A device MAY restrict the length
        and value of this name, possibly space and special characters are not
        allowed.";
      }

      leaf access-control-list-type {
        type access-control-list-type;
        description "Type of access control list. When this
        type is not explicitely specified, if vendor implementation permits,
        the access control entires in the list can be mixed,
        by containing L2, L3 and L4 entries";
      }
```

```
      container access-control-list-oper-data {
        config false;
        description "Overall access control list operational data";

        choice targets{
          description "List of targets where access control list is applied";
          leaf-list interface-name {
            type string;
            description "Interfaces where access control list is applied";
          }
        }
      }

      container access-list-entries {
        description "The access-list-entries container contains
          a list of access-list-entry(ACE).";

        list access-list-entry {
          key rule-name;
          ordered-by user;
          description "List of access list entries(ACE)";
          leaf rule-name {
              type string;
              description "Entry name.";
          }

          container matches {
            description "Define match criteria";
            choice access-list-entries-type {
              description "Type of access list entry.";
              case access-list-entries-ip {
                uses packet-fields:access-control-list-ip-header-fields;
                choice access-list-entries-ip-version {
                  description "Choice of IP version.";
                  case access-list-entries-ipv4 {
                    uses packet-fields:access-control-list-ipv4-header-fields;
                  }
                  case access-list-entries-ipv6 {

                   uses packet-fields:access-control-list-ipv6-header-fields;
                  }
                }
              }
              case access-list-entries-eth {
                description "Ethernet MAC address entry.";
                uses packet-fields:access-control-list-eth-header-fields;
              }
            }
```

```
            uses packet-fields:metadata;
          }

          container actions {
            description "Define action criteria";
            choice packet-handling {
              default deny;

              description "Packet handling action.";
              case deny {
                leaf deny {
                  type empty;
                  description "Deny action.";

                }
              }
              case permit {
                leaf permit {
                  type empty;
                  description "Permit action.";
                }
              }
            }
          }

          container access-list-entries-oper-data {
            config false;

            description "Per access list entries operational data";
            leaf match-counter {
              type yang:counter64;
              description "Number of matches for an access list entry";
            }
          }
        }
      }
    }
  }
}


<CODE ENDS>
```

### 4.2. IETF-PACKET-FIELDS module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, transport layer fields and metadata. These groupings can be augmented to include other proprietary matching criteria. Since the number of match criteria is very large, the base draft does not include these directly but references them by "uses" to keep the base module simple.

```
<CODE BEGINS>file "ietf-packet-fields@2015-03-04.yang"

module ietf-packet-fields {
  yang-version 1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";

  prefix packet-fields;

  import ietf-inet-types {
    prefix "inet";

  }

  import ietf-yang-types {
    prefix "yang";
  }

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

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

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

    WG Chair: Tom Nadeau
    tnadeau@lucidvision.com

    Editor: Dean Bogdanovic
    deanb@juniper.net

    Editor: Kiran Agrahara Sreenivasa
    kkoushik@brocade.com

    Editor: Lisa Huang
```

```
    yihuan@cisco.com

    Editor: Dana Blair

    dblair@cisco.com";

  description
    "This YANG module defines groupings that used by ietf-acl
    but not limited to acl.

    Copyright (c) 2015 IETF Trust and the persons identified as
    the document authors.  All rights reserved.

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

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

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

  revision 2015-03-04 {
    description "Initial version of packet fields used by
      access-lists";
    reference
      "RFC XXXX: Network Access Control List (ACL)
      YANG Data  Model";
  }

  grouping access-control-list-transport-header-fields {
    description "Transport header fields";

    container source-port-range {
      description "inclusive range of source ports";
      leaf lower-port {
        type inet:port-number;
        mandatory true;
        description "Lower boundary.";
      }
      leaf upper-port {
        type inet:port-number;
        description "Upper boundary. If exist, upper port must be greater or
        equal to lower port.";
```

```
      }
    }

    container destination-port-range {
      description "inclusive range of destination ports";
      leaf lower-port {
        type inet:port-number;
        mandatory true;
        description "Lower boundary.";
      }
      leaf upper-port {
        type inet:port-number;
        description "Upper boundary.";
      }
    }
  }

  grouping access-control-list-ip-header-fields {
    description "Header fields common to ipv4 and ipv6";

    uses access-control-list-transport-header-fields;

    leaf dscp {
      type inet:dscp;

      description "Value of dscp.";
    }

    leaf protocol {
      type uint8;
      description "Internet Protocol number.";
    }

  }

  grouping access-control-list-ipv4-header-fields {
    description "fields in IPv4 header";

    leaf destination-ipv4-network {
      type inet:ipv4-prefix;
      description "One or more ip addresses.";
    }

    leaf source-ipv4-network {
      type inet:ipv4-prefix;
      description "One or more ip addresses.";
    }
```

```
  }

  grouping access-control-list-ipv6-header-fields {
    description "fields in IPv6 header";

    leaf destination-ipv6-network {
      type inet:ipv6-prefix;
      description "One or more ip addresses.";
    }

    leaf source-ipv6-network {
      type inet:ipv6-prefix;
      description "One or more ip addresses.";
    }

    leaf flow-label {
      type inet:ipv6-flow-label;
      description "Flow label.";
    }

  }

  grouping access-control-list-eth-header-fields {

    description "fields in ethernet header";

    leaf destination-mac-address {
      type yang:mac-address;
      description "Mac addresses.";
    }

    leaf destination-mac-address-mask {
      type yang:mac-address;
      description "Mac addresses mask.";
    }

    leaf source-mac-address {
      type yang:mac-address;
      description "Mac addresses.";
    }

    leaf source-mac-address-mask {
      type yang:mac-address;
      description "Mac addresses mask.";
    }
  }

  grouping timerange {
```

```
    description "Time range contains time
    segments to allow access-control-list to be
    active/inactive when the system time
    is within the time segments.";

    container absolute {
      description
        "Absolute time and date that
        the associated function starts
        going into effect.";

      leaf start {
        type yang:date-and-time;
        description
        "Start time and date";
      }
      leaf end {
        type yang:date-and-time;
        description "Absolute end time and date";
      }
      leaf active {
        type boolean;
        default "true";
        description

          "Specify the associated function

          active or inactive state when
          starts going into effect";
      }
    } // container absolute
  } //grouping timerange

  grouping metadata {
    description "Fields associated with a packet but not in
      the header";

    leaf input-interface {
      type string;
      description "Packet was received on this interface";
    }
    uses timerange;
  }
}

<CODE ENDS>
```

### 4.3. An ACL Example

Requirement: Deny All traffic from 10.10.10.1 bound for host 10.10.10.255 from leaving.

In order to achieve the requirement, an name access control list is needed. The acl and aces can be described in CLI as the following:

```
access-list ip iacl
deny tcp host 10.10.10.1 host 10.10.10.255
```

Figure 1

Here is the example acl configuration xml:

```
<rpc message-id="101" xmlns:nc="urn:cisco:params:xml:ns:yang:ietf-acl:1.0">
// replace with IANA namespace when assigned
<edit-config>
  <target>
    <running/>
  </target>
  <config>
    <top xmlns="http://example.com/schema/1.2/config">
      <access-lists>
        <access-list>
        <access-control-list-name>sample-ip-acl</access-control-list-name>
        <access-list-entries>
          <access-list-entry>
          <rule-name>telnet-block-rule</rule-name>
          <matches>
            <destination-ipv4-address>10.10.10.255/24</destination-ipv4-address>
            <source-ipv4-address>10.10.10.1/24</source-ipv4-address>
          </matches>
          <actions>
          <deny/>
          </actions>
          </access-list-entry>
        </access-list-entries>
        </access-list>
      </access-lists>
    </top>
  </config>
</edit-config>
</rpc>
```

Figure 2

### 4.4. Port Range Usage Example

When a lower-port and an upper-port are both present, it represents a range between lower-port and upper-port with both the lower-port and upper-port are included. When only a lower-port presents, it represents a single port.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>16387</upper-port>
</source-port-range>
```

This represents source ports 16384,16385, 16386, and 16387.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>65535</upper-port>
</source-port-range>
```

This represents source ports greater than/equal to 16384.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>21</lower-port>
</source-port-range>
```

This represents port 21.

## 5. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used for it (iptables, ip6tables, arptables, ebtables). Recently, this has changed and a single utility, nftables, has been provided. This utility follows very similarly the same base model as proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

## 6. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241] [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242] [RFC6242]. The NETCONF access control model [RFC6536] [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

These are the subtrees and data nodes and their sensitivity/ vulnerability:

/ietf-acl:access-lists/access-list/access-list-entries: This list specifies all the configured access list entries on the device. Unauthorized write access to this list can allow intruders to access and control the system. Unauthorized read access to this list can allow intruders to spoof packets with authorized addresses thereby compromising the system.

## 7. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688] [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

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

URI: urn:ietf:params:xml:ns:yang:ietf-packet-fields

Registrant Contact: The IESG.

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

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-acl prefix: ietf-acl reference: RFC XXXX

name: ietf-packet-fields namespace: urn:ietf:params:xml:ns:yang:ietf-packet-fields prefix: ietf-packet-fields reference: RFC XXXX

## 8. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draft in several past IETF meetings. That draft included an ACL YANG model structure and a rich set of match filters, and acknowledged contributions by Louis Fourie, Dana Blair, Tula Kraiser, Patrick Gili, George Serpa, Martin Bjorklund, Kent Watsen, and Phil Shafer. Many people have reviewed the various earlier drafts that made the draft went into IETF charter.

Dean Bogdanovic, Kiran Agrahara Sreenivasa, Lisa Huang, and Dana Blair each evaluated the YANG model in previous draft separately and then work together, to created a new ACL draft that can be supported by different vendors. The new draft removes vendor specific features, and gives examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with the new one that received more participation from many vendors.

## 9. References

### 9.1. Normative References

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

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

### 9.2. Informative References

[RFC5101] Claise, B., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, January 2008.

## Appendix A. Extending ACL model examples

### A.1. Example of extending existing model for route filtering

With proposed modular design, it is easy to extend the model with other features. Those features can be standard features, like route filters. Route filters match on specific IP addresses or ranges of prefixes. Much like ACLs, they include some match criteria and corresponding match action(s). For that reason, it is very simple to extend existing ACL model with route filtering. The combination of a route prefix and prefix length along with the type of match determines how route filters are evaluated against incoming routes. Different vendors have different match types and in this model we are using only ones that are common across all vendors participating in this draft. As in this example, the base ACL model can be extended with company proprietary extensions, described in the next section.

```
<CODE BEGINS> file "std-ext-route-filter@2015-02-14.yang"

module std-ext-route-filter {
  yang-version 1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-route-filter";

  prefix std-ext-route-filter;

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-acl {
    prefix "ietf-acl";
  }
  organization
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

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

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

  WG Chair: Tom Nadeau
  tnadeau@lucidvision.com

  Editor: Dean Bogdanovic
  deanb@juniper.net
```

```
  Editor: Kiran Agrahara Sreenivasa
  kkoushik@brocade.com

  Editor: Lisa Huang
  yihuan@cisco.com

  Editor: Dana Blair
  dblair@cisco.com";

  description "
    This module describes route filter as a collection of
    match prefixes. When specifying a match prefix, you
    can specify an exact match with a particular route or
    a less precise match. You can configure either a
    common action that applies to the entire list or an
    action associated with each prefix.
    ";

  revision 2015-02-14 {
    description "creating Route-Filter extension model based on ietf-acl model";
    reference " ";
  }

  augment "/ietf-acl:access-lists/ietf-acl:access-list
    /ietf-acl:access-list-entries/
    ietf-acl:access-list-entry/ietf-acl:matches"{
    description "
      This module augments the matches container in the ietf-acl
      module with route filter specific actions
      ";
    choice route-prefix{
      description "Define route filter match criteria";
      case range {
        description "
        Route falls between the lower prefix/prefix-length and the upper
        prefix/prefix-length.
        ";
        choice ipv4-range {
          description "Defines the lower IPv4 prefix/prefix range";
          leaf v4-lower-bound {
            type inet:ipv4-prefix;
            description "Defines the lower IPv4 prefix/prefix length";
          }
          leaf v4-upper-bound {
            type inet:ipv4-prefix;
            description "Defines the upper IPv4 prefix/prefix length";
          }
        }
```

```
        choice ipv6-range {
          description "Defines the IPv6 prefix/prefix range";
          leaf v6-lower-bound {
            type inet:ipv6-prefix;
            description "Defines the lower IPv6 prefix/prefix length";
          }
          leaf v6-upper-bound {
            type inet:ipv6-prefix;
            description "Defines the upper IPv6 prefix/prefix length";
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

## A.2. A company proprietary module example

Module "newco-acl" is an example of company proprietary model that augments "ietf-acl" module. It shows how to use 'augment' with an XPath expression to add additional match criteria, action criteria, and default actions when no ACE matches found. All these are company proprietary extensions or system feature extensions. "newco-acl" is just an example and it is expected from vendors to create their own proprietary models.

The following figure is the tree structure of newco-acl. In this example, ietf-acl:access-lists/ietf-acl:access-list/ietf-acl:access-list-entries/ietf-acl:access-list-entry/ietf-acl:matches: are augmented with a new choice, protocol-payload-choice. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. In other example, ietf-acl:access-lists/ietf-acl :access-list/ietf-acl:access-list-entries/ietf-acl:access-list-entry/ ietf-acl:actions are augmented with new choice of actions.

```
   module: newco-acl
   augment /ietf-acl:access-lists/ietf-acl:access-list
     /ietf-acl:access-list-entries/
     ietf-acl:access-list-entry/ietf-acl:matches:
       +--rw (protocol-payload-choice)?
         +--:(protocol-payload)
           +--rw protocol-payload* [value-keyword]
             +--rw value-keyword    enumeration
   augment /ietf-acl:access-lists/ietf-acl:access-list
     /ietf-acl:access-list-entries/
     ietf-acl:access-list-entry/ietf-acl:actions:
       +--rw (action)?
         +--:(count)
         |  +--rw count?                  string
         +--:(policer)
         |  +--rw policer?                string
         +--:(hiearchical-policer)
           +--rw hierarchitacl-policer?   string
   augment /ietf-acl:access-lists/ietf-acl:access-list:
     +--rw default-actions
       +--rw deny?   empty
```

<CODE BEGINS> file "newco-acl@2015-03-04.yang"

```
module newco-acl {
  yang-version 1;

  namespace "urn:newco:params:xml:ns:yang:newco-acl";

  prefix newco-acl;

  import ietf-acl {
    prefix "ietf-acl";
  }

  revision 2015-03-04{
    description "creating NewCo proprietary extensions to ietf-acl model";
  }

  augment "/ietf-acl:access-lists/ietf-acl:access-list
    /ietf-acl:access-list-entries/
      ietf-acl:access-list-entry/ietf-acl:matches" {
    description "Newco proprietary simple filter matches";
    choice protocol-payload-choice {
      list protocol-payload {
        key value-keyword;
        ordered-by user;
        description "Match protocol payload";
```

```
        uses match-simple-payload-protocol-value;
      }
    }
  }

  augment "/ietf-acl:access-lists/ietf-acl:access-list/ietf-acl:access-list-entr
ies/ietf-acl:access-list-entry/ietf-acl:actions" {
    description "Newco proprietary simple filter actions";
    choice action {
      case count {
        description "Count the packet in the named counter";
        leaf count {
          type string;
        }
      }
      case policer {
        description "Name of policer to use to rate-limit traffic";
        leaf policer {
          type string;
        }
      }
      case hiearchical-policer {
        description "Name of hierarchical policer to use to
        rate-limit traffic";
        leaf hierarchitacl-policer{
          type string;
        }
      }
    }
  }

  augment "/ietf-acl:access-lists/ietf-acl:access-list" {
    container default-actions {
      description "Actions that occur if no access-list entry is matched.";
      leaf deny {
        type empty;
      }
    }
  }

  grouping match-simple-payload-protocol-value {
    leaf value-keyword {
      description "(null)";
      type enumeration {
        enum icmp {
          description "Internet Control Message Protocol";
        }
        enum icmp6 {
          description "Internet Control Message Protocol Version 6";
```

```
        }
        enum range {
          description "Range of values";
        }
      }
    }
  }
}

    <CODE ENDS>
```

Draft authors expect that different vendors will provide their own yang models as in the example above, which is the extension of the base model

## A.3. Attaching Access Control List to interfaces

Access control list typically does not exist in isolation. Instead, they are associated with a certain scope in which they are applied, for example, an interface of a set of interfaces. How to attach an SPF to an interface (or other system artifact) is outside the scope of this model, as it depends on the specifics of the system model that is being applied. However, in general, the general design pattern will involved adding a dada node with a reference, or set of references, to ACLs that are to be applied to the interface. For this purpose, the type definition "access-control-list-ref" can be used.

This is an example of attaching an access control list to an interface.

```
                        <CODE BEGINS> file "interface model augmentation with AC
L
                            @2015-03-04.yang"
  import ietf-acl {
    prefix "ietf-acl";
  }
  import ietf-interface {
    prefix "ietf-if";
  }
  import ietf-yang-types {
    prefix "yang";
  }

  augment "/ietf-if:interfaces/ietf-if:interface" {
    description "Apply acl to interfaces";
    container acl{
      description "ACL related properties.";
      leaf acl-name {
        type ietf-acl:access-control-list-ref;
        mandatory true;
        description "Access Control List name.";
      }
      leaf match-counter {
        type yang:counter64;
        config false;
        description "Total match count for access control list ";
      }
      choice direction {
        leaf in { type empty;}
        leaf out { type empty;}
      }
    }
  }
    <CODE ENDS>
```

Authors' Addresses

Dean Bogdanovic
Juniper Networks

Email: deanb@juniper.net

Kiran Agrahara Sreenivasa
Brocade Communications System

Email: kkoushik@brocade.com

Lisa Huang
Cisco Systems

Email: yihuan@cisco.com

Dana Blair
Cisco Systems

Email: dblair@cisco.com

Network Working Group
Internet-Draft
Obsoletes: 6020 (if approved)
Intended status: Standards Track
Expires: September 10, 2015

M. Bjorklund, Ed.
Tail-f Systems
March 9, 2015

# YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)
draft-ietf-netmod-rfc6020bis-04

## Abstract

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications. This document obsoletes RFC 6020.

## Status of This Memo



## Copyright Notice

Table of Contents

# 1. Introduction

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications. YANG is used to model the operations and content layers of NETCONF (see the NETCONF Configuration Protocol [RFC6241], Section 1.2).

This document describes the syntax and semantics of the YANG language, how the data model defined in a YANG module is represented in the Extensible Markup Language (XML), and how NETCONF operations are used to manipulate the data.

## 1.1. Summary of Changes from RFC 6020

This document defines version 1.1 of the YANG language. YANG version 1.1 is a maintenance release of the YANG language, addressing ambiguities and defects in the original specification [RFC6020].

- o Changed the YANG version from "1" to "1.1".

- o Made noncharacters illegal in the built-in type "string".

- o Defined the legal characters in YANG modules.

- o Made the "yang-version" statement mandatory.

- o Changed the rules for the interpretation of escaped characters in double quoted strings. This is an backwards incompatible change from YANG 1.0. A module that uses a character sequence that is now illegal must change the string to match the new rules. See Section 6.1.3 for details.

- o Extended the "if-feature" syntax to be a boolean expression over feature names.

- o Allow "if-feature" in "bit", "enum", and "identity".

- o Allow "if-feature" in "refine".

- o Made "when" and "if-feature" illegal on list keys, unless the parent is also conditional, and the condition matches the parent’s condition.

- o Allow "choice" as a shorthand case statement (see Section 7.9).

- o Added a new substatement "modifier" to pattern (see Section 9.4.6).

- o Allow "must" in "input", "output", and "notification".

- o Added a set of new XPath functions in Section 10.

- o Clarified the XPath context’s tree in Section 6.4.1.

- o Defined the string value of an identityref in XPath expressions (see Section 9.10).

- o Clarified what unprefixed names means in leafrefs in typedefs (see Section 9.9.2).

- o Allow identities to be derived from multiple base identities (see Section 7.17 and Section 9.10).

- o Allow enumerations to be subtyped (see Section 9.6).

- o Allow leaf-lists to have default values (see Section 7.7.2).

- o Use [RFC7405] syntax for case-sensitive strings in the grammar.

- o Changed the module advertisement mechanism (see Section 5.6.4).

- o Changed the scoping rules for definitions in submodules. A submodule can now reference all defintions in all submodules that belong to the same module, without using the "include" statement.

- o Added a new statement "action" that is used to define operations tied to data nodes.

## 2. Keywords

The keywords "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].

## 3. Terminology

- o action: An operation defined for a node in the data tree.
- o anyxml: A data node that can contain an unknown chunk of XML data.
- o augment: Adds new schema nodes to a previously defined schema node.
- o base type: The type from which a derived type was derived, which may be either a built-in type or another derived type.
- o built-in type: A YANG data type defined in the YANG language, such as uint32 or string.
- o choice: A schema node where only one of a number of identified alternatives is valid.
- o configuration data: The set of writable data that is required to transform a system from its initial default state into its current state [RFC6241].
- o conformance: A measure of how accurately a device follows a data model.
- o container: An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes.
- o data definition statement: A statement that defines new data nodes. One of container, leaf, leaf-list, list, choice, case, augment, uses, and anyxml.
- o data model: A data model describes how data is represented and accessed.
- o data node: A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, and anyxml.

- o data tree: The instantiated tree of configuration and state data on a device.

- o derived type: A type that is derived from a built-in type (such as uint32), or another derived type.

- o device deviation: A failure of the device to implement the module faithfully.

- o extension: An extension attaches non-YANG semantics to statements. The extension statement defines new statements to express these semantics.

- o feature: A mechanism for marking a portion of the model as optional. Definitions can be tagged with a feature name and are only valid on devices that support that feature.

- o grouping: A reusable set of schema nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it. The grouping statement is not a data definition statement and, as such, does not define any nodes in the schema tree.

- o identifier: Used to identify different kinds of YANG items by name.

- o instance identifier: A mechanism for identifying a particular node in a data tree.

- o interior node: Nodes within a hierarchy that are not leaf nodes.

- o leaf: A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.

- o leaf-list: Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.

- o list: An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.

- o module: A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and "compilable".

- o RPC: A Remote Procedure Call, as used within the NETCONF protocol.

- o RPC operation: A specific Remote Procedure Call, as used within the NETCONF protocol. It is also called a protocol operation.

- o schema node: A node in the schema tree. One of container, leaf, leaf-list, list, choice, case, rpc, input, output, notification, and anyxml.

- o schema node identifier: A mechanism for identifying a particular node in the schema tree.

- o schema tree: The definition hierarchy specified within a module.

- o state data: The additional data on a system that is not configuration data such as read-only status information and collected statistics [RFC6241].

- o submodule: A partial module definition that contributes derived types, groupings, data nodes, RPCs, and notifications to a module. A YANG module can be constructed from a number of submodules.

- o top-level data node: A data node where there is no other data node between it and a module or submodule statement.

- o uses: The "uses" statement is used to instantiate the set of schema nodes defined in a grouping statement. The instantiated nodes may be refined and augmented to tailor them to any specific needs.

### 3.1. Mandatory Nodes

A mandatory node is one of:

- o A leaf, choice, or anyxml node with a "mandatory" statement with the value "true".

- o A list or leaf-list node with a "min-elements" statement with a value greater than zero.

- o A container node without a "presence" statement, which has at least one mandatory node as a child.

## 4. YANG Overview

### 4.1. Functional Overview

YANG is a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, Remote

Procedure Calls (RPCs), and notifications. This allows a complete description of all data sent between a NETCONF client and server.

YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between those nodes.

YANG structures data models into modules and submodules. A module can import data from other external modules, and include data from submodules. The hierarchy can be augmented, allowing one module to add data nodes to the hierarchy defined in another module. This augmentation can be conditional, with new nodes appearing only if certain conditions are met.

YANG models can describe constraints to be enforced on the data, restricting the appearance or value of nodes based on the presence or value of other nodes in the hierarchy. These constraints are enforceable by either the client or the server, and valid content MUST abide by them.

YANG defines a set of built-in types, and has a type mechanism through which additional types may be defined. Derived types can restrict their base type’s set of valid values using mechanisms like range or pattern restrictions that can be enforced by clients or servers. They can also define usage conventions for use of the derived type, such as a string-based type that contains a host name.

YANG permits the definition of reusable groupings of nodes. The instantiation of these groupings can refine or augment the nodes, allowing it to tailor the nodes to its particular needs. Derived types and groupings can be defined in one module or submodule and used in either that location or in another module or submodule that imports or includes it.

YANG data hierarchy constructs include defining lists where list entries are identified by keys that distinguish them from each other. Such lists may be defined as either sorted by user or automatically sorted by the system. For user-sorted lists, operations are defined for manipulating the order of the list entries.

YANG modules can be translated into an equivalent XML syntax called YANG Independent Notation (YIN) (Section 12), allowing applications using XML parsers and Extensible Stylesheet Language Transformations (XSLT) scripts to operate on the models. The conversion from YANG to YIN is lossless, so content in YIN can be round-tripped back into YANG.

YANG strikes a balance between high-level data modeling and low-level bits-on-the-wire encoding. The reader of a YANG module can see the high-level view of the data model while understanding how the data will be encoded in NETCONF operations.

YANG is an extensible language, allowing extension statements to be defined by standards bodies, vendors, and individuals. The statement syntax allows these extensions to coexist with standard YANG statements in a natural way, while extensions in a YANG module stand out sufficiently for the reader to notice them.

YANG resists the tendency to solve all possible problems, limiting the problem space to allow expression of NETCONF data models, not arbitrary XML documents or arbitrary data models. The data models described by YANG are designed to be easily operated upon by NETCONF operations.

To the extent possible, YANG maintains compatibility with Simple Network Management Protocol's (SNMP's) SMIv2 (Structure of Management Information version 2 [RFC2578], [RFC2579]). SMIv2-based MIB modules can be automatically translated into YANG modules for read-only access. However, YANG is not concerned with reverse translation from YANG to SMIv2.

Like NETCONF, YANG targets smooth integration with the device's native management infrastructure. This allows implementations to leverage their existing access control mechanisms to protect or expose elements of the data model.

## 4.2. Language Overview

This section introduces some important constructs used in YANG that will aid in the understanding of the language specifics in later sections. This progressive approach handles the inter-related nature of YANG concepts and statements. A detailed description of YANG statements and syntax begins in Section 7.

### 4.2.1. Modules and Submodules

A module contains three types of statements: module-header statements, revision statements, and definition statements. The module header statements describe the module and give information about the module itself, the revision statements give information about the history of the module, and the definition statements are the body of the module where the data model is defined.

A NETCONF server may implement a number of modules, allowing multiple views of the same data, or multiple views of disjoint subsections of

the device’s data. Alternatively, the server may implement only one module that defines all available data.

A module may be divided into submodules, based on the needs of the module owner. The external view remains that of a single module, regardless of the presence or size of its submodules.

The "import" statement allows a module or submodule to reference material defined in other modules.

The "include" statement is used by a module to incorporate the contents of its submodules into the module.

#### 4.2.2. Data Modeling Basics

YANG defines four types of nodes for data modeling. In each of the following subsections, the example shows the YANG syntax as well as a corresponding NETCONF XML representation.

##### 4.2.2.1. Leaf Nodes

A leaf node contains simple data like an integer or a string. It has exactly one value of a particular type and no child nodes.

YANG Example:

```
leaf host-name {
    type string;
    description "Hostname for this system";
}
```

NETCONF XML Example:

```
<host-name>my.example.com</host-name>
```

The "leaf" statement is covered in Section 7.6.

##### 4.2.2.2. Leaf-List Nodes

A leaf-list defines a sequence of values of a particular type.

YANG Example:

```
leaf-list domain-search {
    type string;
    description "List of domain names to search";
}
```

NETCONF XML Example:

```
<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>everywhere.example.com</domain-search>
```

The "leaf-list" statement is covered in Section 7.7.

#### 4.2.2.3. Container Nodes

A container node is used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).

YANG Example:

```
container system {
    container login {
        leaf message {
            type string;
            description
                "Message given at start of login session";
        }
    }
}
```

NETCONF XML Example:

```
<system>
  <login>
    <message>Good morning</message>
  </login>
</system>
```

The "container" statement is covered in Section 7.5.

#### 4.2.2.4. List Nodes

A list defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely identified by the values of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type (including leafs, lists, containers etc.).

YANG Example:

```
list user {
    key "name";
    leaf name {
        type string;
    }
    leaf full-name {
        type string;
    }
    leaf class {
        type string;
    }
}
```

NETCONF XML Example:

```
<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>
```

The "list" statement is covered in Section 7.8.

#### 4.2.2.5. Example Module

These statements are combined to define the module:

```
// Contents of "acme-system.yang"
module acme-system {
    yang-version 1.1;
    namespace "http://acme.example.com/system";
    prefix "acme";

    organization "ACME Inc.";
    contact "joe@acme.example.com";
    description
        "The module for entities implementing the ACME system.";

    revision 2007-06-09 {
        description "Initial revision.";
    }

    container system {
        leaf host-name {
            type string;
            description "Hostname for this system";
        }

        leaf-list domain-search {
            type string;
            description "List of domain names to search";
        }

        container login {
            leaf message {
                type string;
                description
                    "Message given at start of login session";
            }

            list user {
                key "name";
                leaf name {
                    type string;
                }
                leaf full-name {
                    type string;
                }
                leaf class {
                    type string;
                }
            }
        }
    }
}
```

### 4.2.3. State Data

YANG can model state data, as well as configuration data, based on the "config" statement. When a node is tagged with "config false", its subhierarchy is flagged as state data, to be reported using NETCONF's <get> operation, not the <get-config> operation. Parent containers, lists, and key leafs are reported also, giving the context for the state data.

In this example, two leafs are defined for each interface, a configured speed and an observed speed. The observed speed is not configuration, so it can be returned with NETCONF <get> operations, but not with <get-config> operations. The observed speed is not configuration data, and it cannot be manipulated using <edit-config>.

```
list interface {
    key "name";

    leaf name {
        type string;
    }
    leaf speed {
        type enumeration {
            enum 10m;
            enum 100m;
            enum auto;
        }
    }
    leaf observed-speed {
        type uint32;
        config false;
    }
}
```

### 4.2.4. Built-In Types

YANG has a set of built-in types, similar to those of many programming languages, but with some differences due to special requirements from the management domain. The following table summarizes the built-in types discussed in Section 9:

| Name | Description |
|---|---|
| binary | Any binary data |
| bits | A set of bits or flags |
| boolean | "true" or "false" |
| decimal64 | 64-bit signed decimal number |
| empty | A leaf that does not have any value |
| enumeration | Enumerated strings |
| identityref | A reference to an abstract identity |
| instance-identifier | References a data tree node |
| int8 | 8-bit signed integer |
| int16 | 16-bit signed integer |
| int32 | 32-bit signed integer |
| int64 | 64-bit signed integer |
| leafref | A reference to a leaf instance |
| string | Human-readable string |
| uint8 | 8-bit unsigned integer |
| uint16 | 16-bit unsigned integer |
| uint32 | 32-bit unsigned integer |
| uint64 | 64-bit unsigned integer |
| union | Choice of member types |

The "type" statement is covered in Section 7.4.

#### 4.2.5. Derived Types (typedef)

YANG can define derived types from base types using the "typedef" statement. A base type can be either a built-in type or a derived type, allowing a hierarchy of derived types.

A derived type can be used as the argument for the "type" statement.

YANG Example:

```
typedef percent {
    type uint8 {
        range "0 .. 100";
    }
    description "Percentage";
}

leaf completed {
    type percent;
}
```

NETCONF XML Example:

```
<completed>20</completed>
```

The "typedef" statement is covered in Section 7.3.

#### 4.2.6. Reusable Node Groups (grouping)

Groups of nodes can be assembled into reusable collections using the "grouping" statement. A grouping defines a set of nodes that are instantiated with the "uses" statement:

```
grouping target {
    leaf address {
        type inet:ip-address;
        description "Target IP address";
    }
    leaf port {
        type inet:port-number;
        description "Target port number";
    }
}

container peer {
    container destination {
        uses target;
    }
}
```

NETCONF XML Example:

```
<peer>
  <destination>
    <address>192.0.2.1</address>
    <port>830</port>
  </destination>
</peer>
```

The grouping can be refined as it is used, allowing certain statements to be overridden. In this example, the description is refined:

```
container connection {
    container source {
        uses target {
            refine "address" {
                description "Source IP address";
            }
            refine "port" {
                description "Source port number";
            }
        }
    }
    container destination {
        uses target {
            refine "address" {
                description "Destination IP address";
            }
            refine "port" {
                description "Destination port number";
            }
        }
    }
}
```

The "grouping" statement is covered in Section 7.11.

#### 4.2.7. Choices

YANG allows the data model to segregate incompatible nodes into distinct choices using the "choice" and "case" statements. The "choice" statement contains a set of "case" statements that define sets of schema nodes that cannot appear together. Each "case" may contain multiple nodes, but each node may appear in only one "case" under a "choice".

When a node from one case is created in the data tree, all nodes from all other cases are implicitly deleted. The device handles the enforcement of the constraint, preventing incompatibilities from existing in the configuration.

The choice and case nodes appear only in the schema tree, not in the data tree or NETCONF messages. The additional levels of hierarchy are not needed beyond the conceptual schema.

YANG Example:

```
     container food {
       choice snack {
           case sports-arena {
               leaf pretzel {
                   type empty;
               }
               leaf beer {
                   type empty;
               }
           }
           case late-night {
               leaf chocolate {
                   type enumeration {
                       enum dark;
                       enum milk;
                       enum first-available;
                   }
               }
           }
       }
    }
```

NETCONF XML Example:

```
     <food>
       <pretzel/>
       <beer/>
     </food>
```

The "choice" statement is covered in Section 7.9.

#### 4.2.8. Extending Data Models (augment)

YANG allows a module to insert additional nodes into data models, including both the current module (and its submodules) or an external module. This is useful for example for vendors to add vendor-specific parameters to standard data models in an interoperable way.

The "augment" statement defines the location in the data model hierarchy where new nodes are inserted, and the "when" statement defines the conditions when the new nodes are valid.

YANG Example:

```
augment /system/login/user {
    when "class != 'wheel'";
    leaf uid {
        type uint16 {
            range "1000 .. 30000";
        }
    }
}
```

This example defines a "uid" node that only is valid when the user's "class" is not "wheel".

If a module augments another module, the XML representation of the data will reflect the prefix of the augmenting module. For example, if the above augmentation were in a module with prefix "other", the XML would look like:

NETCONF XML Example:

```
<user>
  <name>alicew</name>
  <full-name>Alice N. Wonderland</full-name>
  <class>drop-out</class>
  <other:uid>1024</other:uid>
</user>
```

The "augment" statement is covered in Section 7.16.

#### 4.2.9. Operation Definitions

YANG allows the definition of operations. The operations' names, input parameters, and output parameters are modeled using YANG data definition statements. Operations on the top-level in a module are defined with the "rpc" statement. Operations can also be tied to a node in the data hierarchy. Such operations are defined with the "action" statement.

YANG Example:

```
rpc activate-software-image {
    input {
        leaf image-name {
            type string;
        }
    }
    output {
        leaf status {
            type string;
        }
    }
}
```

NETCONF XML Example:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <activate-software-image xmlns="http://acme.example.com/system">
    <image-name>acmefw-2.3</image-name>
 </activate-software-image>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <status xmlns="http://acme.example.com/system">
    The image acmefw-2.3 is being installed.
  </status>
</rpc-reply>
```

The "rpc" statement is covered in Section 7.13, and the "action" statement in Section 7.14.

#### 4.2.10. Notification Definitions

YANG allows the definition of notifications suitable for NETCONF. YANG data definition statements are used to model the content of the notification.

YANG Example:

```
notification link-failure {
    description "A link failure has been detected";
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf if-admin-status {
        type admin-status;
    }
    leaf if-oper-status {
        type oper-status;
    }
}
```

NETCONF XML Example:

```
<notification
    xmlns="urn:ietf:params:netconf:capability:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

The "notification" statement is covered in Section 7.15.

# 5. Language Concepts

## 5.1. Modules and Submodules

The module is the base unit of definition in YANG. A module defines a single data model. A module can define a complete, cohesive model, or augment an existing data model with additional nodes.

Submodules are partial modules that contribute definitions to a module. A module may include any number of submodules, but each submodule may belong to only one module.

The names of all standard modules and submodules MUST be unique. Developers of enterprise modules are RECOMMENDED to choose names for their modules that will have a low probability of colliding with standard or other enterprise modules, e.g., by using the enterprise or organization name as a prefix for the module name.

A module uses the "include" statement to include all its submodules, and the "import" statement to reference external modules. Similarly, a submodule uses the "import" statement to reference other modules.

For backwards compatibility with YANG version 1, a submodule is allowed it use the "include" statement to reference other submodules within its module, but this is not necessary in YANG version 1.1. A submodule can reference any definition in the module it belongs to and in all submodules included by the module.

A module or submodule MUST NOT include submodules from other modules, and a submodule MUST NOT import its own module.

The import and include statements are used to make definitions available from other modules:

- For a module or submodule to reference definitions in an external module, the external module MUST be imported.

- A module MUST include all its submodules.

- A module or submodule belonging to that module can reference definitions in the module and all submodules included by the module.

There MUST NOT be any circular chains of imports or includes. For example, if module "a" imports module "b", "b" cannot import "a".

When a definition in an external module is referenced, a locally defined prefix MUST be used, followed by ":", and then the external identifier. References to definitions in the local module MAY use the prefix notation. Since built-in data types do not belong to any module and have no prefix, references to built-in data types (e.g., int32) cannot use the prefix notation. The syntax for a reference to a definition is formally defined by the rule "identifier-ref" in Section 13.

### 5.1.1. Import and Include by Revision

Published modules evolve independently over time. In order to allow for this evolution, modules need to be imported using specific revisions. When a module is written, it uses the current revisions of other modules, based on what is available at the time. As future revisions of the imported modules are published, the importing module is unaffected and its contents are unchanged. When the author of the module is prepared to move to the most recently published revision of an imported module, the module is republished with an updated "import" statement. By republishing with the new revision, the

authors explicitly indicate their acceptance of any changes in the imported module.

For submodules, the issue is related but simpler. A module or submodule that includes submodules needs to specify the revision of the included submodules. If a submodule changes, any module or submodule that includes it needs to be updated.

For example, module "b" imports module "a".

```
module a {
    yang-version 1.1;
    namespace "http://example.com/a";
    prefix "a";

    revision 2008-01-01 { ... }
    grouping a {
        leaf eh { .... }
    }
}

module b {
    yang-version 1.1;
    namespace "http://example.com/b";
    prefix "b";

    import a {
        prefix p;
        revision-date 2008-01-01;
    }

    container bee {
        uses p:a;
    }
}
```

When the author of "a" publishes a new revision, the changes may not be acceptable to the author of "b". If the new revision is acceptable, the author of "b" can republish with an updated revision in the "import" statement.

### 5.1.2. Module Hierarchies

YANG allows modeling of data in multiple hierarchies, where data may have more than one top-level node. Models that have multiple top-level nodes are sometimes convenient, and are supported by YANG.

NETCONF is capable of carrying any XML content as the payload in the <config> and <data> elements. The top-level nodes of YANG modules are encoded as child elements, in any order, within these elements. This encapsulation guarantees that the corresponding NETCONF messages are always well-formed XML documents.

For example:

```
module my-config {
    yang-version 1.1;
    namespace "http://example.com/schema/config";
    prefix "co";

    container system { ... }
    container routing { ... }
}
```

could be encoded in NETCONF as:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <!-- system data here -->
      </system>
      <routing xmlns="http://example.com/schema/config">
        <!-- routing data here -->
      </routing>
    </config>
  </edit-config>
</rpc>
```

## 5.2. File Layout

YANG modules and submodules are typically stored in files, one module or submodule per file. The name of the file SHOULD be of the form:

```
module-or-submodule-name ['@' revision-date] ( '.yang' / '.yin' )
```

YANG compilers can find imported modules and included submodules via this convention. While the YANG language defines modules, tools may compile submodules independently for performance and manageability reasons. Errors and warnings that cannot be detected during

submodule compilation may be delayed until the submodules are linked into a cohesive module.

## 5.3. XML Namespaces

All YANG definitions are specified within a module that is bound to a particular XML namespace [XML-NAMES], which is a globally unique URI [RFC3986]. A NETCONF client or server uses the namespace during XML encoding of data.

Namespaces for modules published in RFC streams [RFC4844] MUST be assigned by IANA, see Section 15.

Namespaces for private modules are assigned by the organization owning the module without a central registry. Namespace URIs MUST be chosen so they cannot collide with standard or other enterprise namespaces, for example by using the enterprise or organization name in the namespace.

The "namespace" statement is covered in Section 7.1.3.

### 5.3.1. YANG XML Namespace

YANG defines an XML namespace for NETCONF <edit-config> operations, <error-info> content, and the <action> element. The name of this namespace is "urn:ietf:params:xml:ns:yang:1".

## 5.4. Resolving Grouping, Type, and Identity Names

Grouping, type, and identity names are resolved in the context in which they are defined, rather than the context in which they are used. Users of groupings, typedefs, and identities are not required to import modules or include submodules to satisfy all references made by the original definition. This behaves like static scoping in a conventional programming language.

For example, if a module defines a grouping in which a type is referenced, when the grouping is used in a second module, the type is resolved in the context of the original module, not the second module. There is no worry over conflicts if both modules define the type, since there is no ambiguity.

## 5.5. Nested Typedefs and Groupings

Typedefs and groupings may appear nested under many YANG statements, allowing these to be lexically scoped by the hierarchy under which they appear. This allows types and groupings to be defined near

where they are used, rather than placing them at the top level of the hierarchy. The close proximity increases readability.

Scoping also allows types to be defined without concern for naming conflicts between types in different submodules. Type names can be specified without adding leading strings designed to prevent name collisions within large modules.

Finally, scoping allows the module author to keep types and groupings private to their module or submodule, preventing their reuse. Since only top-level types and groupings (i.e., those appearing as substatements to a module or submodule statement) can be used outside the module or submodule, the developer has more control over what pieces of their module are presented to the outside world, supporting the need to hide internal information and maintaining a boundary between what is shared with the outside world and what is kept private.

Scoped definitions MUST NOT shadow definitions at a higher scope. A type or grouping cannot be defined if a higher level in the schema hierarchy has a definition with a matching identifier.

A reference to an unprefixed type or grouping, or one which uses the prefix of the current module, is resolved by locating the closest matching "typedef" or "grouping" statement among the immediate substatements of each ancestor statement.

## 5.6. Conformance

Conformance is a measure of how accurately a device follows the model. Generally speaking, devices are responsible for implementing the model faithfully, allowing applications to treat devices which implement the model identically. Deviations from the model can reduce the utility of the model and increase fragility of applications that use it.

YANG modelers have three mechanisms for conformance:

- the basic behavior of the model
- optional features that are part of the model
- deviations from the model

We will consider each of these in sequence.

### 5.6.1. Basic Behavior

The model defines a contract between the NETCONF client and server, which allows both parties to have faith the other knows the syntax and semantics behind the modeled data. The strength of YANG lies in the strength of this contract.

### 5.6.2. Optional Features

In many models, the modeler will allow sections of the model to be conditional. The device controls whether these conditional portions of the model are supported or valid for that particular device.

For example, a syslog data model may choose to include the ability to save logs locally, but the modeler will realize that this is only possible if the device has local storage. If there is no local storage, an application should not tell the device to save logs.

YANG supports this conditional mechanism using a construct called "feature". Features give the modeler a mechanism for making portions of the module conditional in a manner that is controlled by the device. The model can express constructs that are not universally present in all devices. These features are included in the model definition, allowing a consistent view and allowing applications to learn which features are supported and tailor their behavior to the device.

A module may declare any number of features, identified by simple strings, and may make portions of the module optional based on those features. If the device supports a feature, then the corresponding portions of the module are valid for that device. If the device doesn’t support the feature, those parts of the module are not valid, and applications should behave accordingly.

Features are defined using the "feature" statement. Definitions in the module that are conditional to the feature are noted by the "if-feature" statement.

Further details are available in Section 7.19.1.

### 5.6.3. Deviations

In an ideal world, all devices would be required to implement the model exactly as defined, and deviations from the model would not be allowed. But in the real world, devices are often not able or designed to implement the model as written. For YANG-based automation to deal with these device deviations, a mechanism must

exist for devices to inform applications of the specifics of such deviations.

For example, a BGP module may allow any number of BGP peers, but a particular device may only support 16 BGP peers. Any application configuring the 17th peer will receive an error. While an error may suffice to let the application know it cannot add another peer, it would be far better if the application had prior knowledge of this limitation and could prevent the user from starting down the path that could not succeed.

Device deviations are declared using the "deviation" statement, which takes as its argument a string that identifies a node in the schema tree. The contents of the statement details the manner in which the device implementation deviates from the contract as defined in the module.

Further details are available in Section 7.19.3.

### 5.6.4. Announcing Conformance Information in the <hello> Message

This document defines the following mechanism for announcing conformance information. Other mechanisms may be defined by future specificiations.

A NETCONF server announces the modules it implements by implementing the YANG module "ietf-yang-library" defined in [I-D.ietf-netconf-yang-library]. The server also advertises the following capability in the <hello> message (line-breaks and whitespaces are used for formatting reasons only):

```
urn:ietf:params:netconf:capability:yang-library:1.0?
  module-set-id=<id>
```

The parameter "module-set-id" has the same value as the leaf "/modules/module-set-id" from "ietf-yang-library". This parameter MUST be present.

With this mechanism, a client can cache the supported modules for a server, and only update the cache if the "module-set-id" value in the <hello> message changes.

## 5.7. Data Store Modification

Data models may allow the server to alter the configuration data store in ways not explicitly directed via NETCONF protocol messages. For example, a data model may define leafs that are assigned system-generated values when the client does not provide one. A formal

mechanism for specifying the circumstances where these changes are allowed is out of scope for this specification.

# 6. YANG Syntax

The YANG syntax is similar to that of SMIng [RFC3780] and programming languages like C and C++. This C-like syntax was chosen specifically for its readability, since YANG values the time and effort of the readers of models above those of modules writers and YANG tool-chain developers. This section introduces the YANG syntax.

YANG modules use the UTF-8 [RFC3629] character encoding.

Legal characters in YANG modules are the Unicode and ISO/IEC 10646 [ISO.10646] characters, including tab, carriage return, and line feed but excluding the other C0 control characters, the surrogate blocks, and the noncharacters. The character syntax is formally defined by the rule "yang-char" in Section 13.

## 6.1. Lexical Tokenization

YANG modules are parsed as a series of tokens. This section details the rules for recognizing tokens from an input stream. YANG tokenization rules are both simple and powerful. The simplicity is driven by a need to keep the parsers easy to implement, while the power is driven by the fact that modelers need to express their models in readable formats.

### 6.1.1. Comments

Comments are C++ style. A single line comment starts with "//" and ends at the end of the line. A block comment is enclosed within "/*" and "*/".

### 6.1.2. Tokens

A token in YANG is either a keyword, a string, a semicolon (";"), or braces ("{" or "}"). A string can be quoted or unquoted. A keyword is either one of the YANG keywords defined in this document, or a prefix identifier, followed by ":", followed by a language extension keyword. Keywords are case sensitive. See Section 6.2 for a formal definition of identifiers.

### 6.1.3. Quoting

If a string contains any space or tab characters, a semicolon (";"), braces ("{" or "}"), or comment sequences ("//", "/*", or "*/"), then it MUST be enclosed within double or single quotes.

If the double-quoted string contains a line break followed by space or tab characters that are used to indent the text according to the layout in the YANG file, this leading whitespace is stripped from the string, up to and including the column of the double quote character, or to the first non-whitespace character, whichever occurs first. In this process, a tab character is treated as 8 space characters.

If the double-quoted string contains space or tab characters before a line break, this trailing whitespace is stripped from the string.

A single-quoted string (enclosed within ’ ’) preserves each character within the quotes. A single quote character cannot occur in a single-quoted string, even when preceded by a backslash.

Within a double-quoted string (enclosed within " "), a backslash character introduces a special character, which depends on the character that immediately follows the backslash:

```
 \n      new line
 \t      a tab character
 \"      a double quote
 \\      a single backslash
```

It is an error if any other character follows the backslash character.

If a quoted string is followed by a plus character ("+"), followed by another quoted string, the two strings are concatenated into one string, allowing multiple concatenations to build one string. Whitespace trimming and substitution of backslash-escaped characters in double-quoted strings is done before concatenation.

#### 6.1.3.1. Quoting Examples

The following strings are equivalent:

```
  hello
  "hello"
  ’hello’
  "hel" + "lo"
  ’hel’ + "lo"
```

The following examples show some special strings:

```
"\""  - string containing a double quote
'"'   - string containing a double quote
"\n"  - string containing a new line character
'\n'  - string containing a backslash followed
        by the character n
```

The following examples show some illegal strings:

```
''''  - a single-quoted string cannot contain single quotes
"""   - a double quote must be escaped in a double-quoted string
```

The following strings are equivalent:

```
      "first line
         second line"

"first line\n" + "  second line"
```

## 6.2. Identifiers

Identifiers are used to identify different kinds of YANG items by name. Each identifier starts with an uppercase or lowercase ASCII letter or an underscore character, followed by zero or more ASCII letters, digits, underscore characters, hyphens, and dots. Implementations MUST support identifiers up to 64 characters in length. Identifiers are case sensitive. The identifier syntax is formally defined by the rule "identifier" in Section 13. Identifiers can be specified as quoted or unquoted strings.

### 6.2.1. Identifiers and Their Namespaces

Each identifier is valid in a namespace that depends on the type of the YANG item being defined. All identifiers defined in a namespace MUST be unique.

- o All module and submodule names share the same global module identifier namespace.

- o All extension names defined in a module and its submodules share the same extension identifier namespace.

- o All feature names defined in a module and its submodules share the same feature identifier namespace.

- o All identity names defined in a module and its submodules share the same identity identifier namespace.

- o All derived type names defined within a parent node or at the top level of the module or its submodules share the same type identifier namespace. This namespace is scoped to all descendant nodes of the parent node or module. This means that any descendent node may use that typedef, and it MUST NOT define a typedef with the same name.

- o All grouping names defined within a parent node or at the top level of the module or its submodules share the same grouping identifier namespace. This namespace is scoped to all descendant nodes of the parent node or module. This means that any descendent node may use that grouping, and it MUST NOT define a grouping with the same name.

- o All leafs, leaf-lists, lists, containers, choices, rpcs, notifications, and anyxmls defined (directly or through a uses statement) within a parent node or at the top level of the module or its submodules share the same identifier namespace. This namespace is scoped to the parent node or module, unless the parent node is a case node. In that case, the namespace is scoped to the closest ancestor node that is not a case or choice node.

- o All cases within a choice share the same case identifier namespace. This namespace is scoped to the parent choice node.

Forward references are allowed in YANG.

## 6.3. Statements

A YANG module contains a sequence of statements. Each statement starts with a keyword, followed by zero or one argument, followed either by a semicolon (";") or a block of substatements enclosed within braces ("{ }"):

```
statement = keyword [argument] (";" / "{" *statement "}")
```

The argument is a string, as defined in Section 6.1.2.

### 6.3.1. Language Extensions

A module can introduce YANG extensions by using the "extension" keyword (see Section 7.18). The extensions can be imported by other modules with the "import" statement (see Section 7.1.5). When an imported extension is used, the extension’s keyword MUST be qualified using the prefix with which the extension’s module was imported. If an extension is used in the module where it is defined, the extension’s keyword MUST be qualified with the module’s prefix.

If a YANG compiler does not support a particular extension, which appears in a YANG module as an unknown-statement (see Section 13), the entire unknown-statement MAY be ignored by the compiler.

### 6.4. XPath Evaluations

YANG relies on XML Path Language (XPath) 1.0 [XPATH] as a notation for specifying many inter-node references and dependencies. NETCONF clients and servers are not required to implement an XPath interpreter, but MUST ensure that the requirements encoded in the data model are enforced. The manner of enforcement is an implementation decision. The XPath expressions MUST be syntactically correct, and all prefixes used MUST be present in the XPath context (see Section 6.4.1). An implementation may choose to implement them by hand, rather than using the XPath expression directly.

The data model used in the XPath expressions is the same as that used in XPath 1.0 [XPATH], with the same extension for root node children as used by XSLT 1.0 [XSLT] (Section 3.1). Specifically, it means that the root node may have any number of element nodes as its children.

Numbers in XPath 1.0 are IEEE 754 double-precision floating-point values, see Section 3.5 in [XPATH]. This means that some values of int64, uint64 and decimal64 types (see Section 9.2 and Section 9.3) cannot be exactly represented in XPath expressions. Therefore, due caution should be exercised when using nodes with 64-bit numeric values in XPath expressions. In particular, numerical comparisons involving equality may yield unexpected results.

For example, consider the following definition:

```
leaf lxiv {
    type decimal64 {
        fraction-digits 18;
    }
    must ". <= 10";
}
```

An instance of the "lxiv" leaf having the value of 10.0000000000000001 will then successfully pass validation.

#### 6.4.1. XPath Context

All YANG XPath expressions share the following XPath context definition:

- o The set of namespace declarations is the set of all "import" statements' prefix and namespace pairs in the module where the XPath expression is specified, and the "prefix" statement's prefix for the "namespace" statement's URI.

- o Names without a namespace prefix belong to the same namespace as the identifier of the current node. Inside a grouping, that namespace is affected by where the grouping is used (see Section 7.12). Inside a typedef, that namespace is affected by where the typedef is referenced. If a typedef is defined and referenced within a grouping, the namespace is affected by where the grouping is used (see Section 7.12).

- o The function library is the core function library defined in [XPATH], and the functions defined in Section 10.

- o The set of variable bindings is empty.

The mechanism for handling unprefixed names is adopted from XPath 2.0 [XPATH2.0], and helps simplify XPath expressions in YANG. No ambiguity may ever arise because YANG node identifiers are always qualified names with a non-null namespace URI.

The accessible tree depends on where the statement with the XPath expression is defined:

- o If the XPath expression is defined in substatement to a data node that represents configuration, the accessible tree is the data in the NETCONF datastore where the context node exists. The root node has all top-level configuration data nodes in all modules as children.

- o If the XPath expression is defined in a substatement to a data node that represents state data, the accessible tree is all all state data on the device, and the "running" datastore. The root node has all top-level data nodes in all modules as children.

- o If the XPath expression is defined in a substatement to a "notification" statement, the accessible tree is the notification instance, all state data on the device, and the "running" datastore. The root node has the node representing the notification being defined and all top-level data nodes in all modules as children.

- o If the XPath expression is defined in a substatement to an "input" statement in an "rpc" statement, the accessible tree is the RPC operation instance, all state data on the device, and the "running" datastore. The root node has the node representing the

operation being defined and all top-level data nodes in all modules as children. The node representing the operation being defined has the operation’s input parameters as children.

- o If the XPath expression is defined in a substatement to an "output" statement in an "rpc" statement, the accessible tree is the RPC operation’s output, all state data on the device, and the "running" datastore. The root node has the node representing the operation being defined and all top-level data nodes in all modules as children. The node representing the operation being defined has the operation’s output parameters as children.

In the accessible tree, all leafs and leaf-lists with default values in use exist (See Section 7.6.1 and Section 7.7.2).

If a node that exists in the accessible tree has a non-presence container as a child, then the non-presence container also exists in the tree.

The context node varies with the YANG XPath expression, and is specified where the YANG statement with the XPath expression is defined.

## 6.5. Schema Node Identifier

A schema node identifier is a string that identifies a node in the schema tree. It has two forms, "absolute" and "descendant", defined by the rules "absolute-schema-nodeid" and "descendant-schema-nodeid" in Section 13, respectively. A schema node identifier consists of a path of identifiers, separated by slashes ("/"). In an absolute schema node identifier, the first identifier after the leading slash is any top-level schema node in the local module or in all imported modules.

References to identifiers defined in external modules MUST be qualified with appropriate prefixes, and references to identifiers defined in the current module and its submodules MAY use a prefix.

For example, to identify the child node "b" of top-level node "a", the string "/a/b" can be used.

# 7. YANG Statements

The following sections describe all of the YANG statements.

Note that even a statement that does not have any substatements defined in YANG can have vendor-specific extensions as substatements.

For example, the "description" statement does not have any substatements defined in YANG, but the following is legal:

```
description "some text" {
    acme:documentation-flag 5;
}
```

## 7.1. The module Statement

The "module" statement defines the module’s name, and groups all statements that belong to the module together. The "module" statement’s argument is the name of the module, followed by a block of substatements that hold detailed module information. The module name follows the rules for identifiers in Section 6.2.

Names of modules published in RFC streams [RFC4844] MUST be assigned by IANA, see Section 15.

Private module names are assigned by the organization owning the module without a central registry. It is RECOMMENDED to choose module names that will have a low probability of colliding with standard or other enterprise modules and submodules, e.g., by using the enterprise or organization name as a prefix for the module name.

A module typically has the following layout:

```
module <module-name> {

    // header information
    <yang-version statement>
    <namespace statement>
    <prefix statement>

    // linkage statements
    <import statements>
    <include statements>

    // meta information
    <organization statement>
    <contact statement>
    <description statement>
    <reference statement>

    // revision history
    <revision statements>

    // module definitions
    <other statements>
}
```

#### 7.1.1. The module’s Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| augment | 7.16 | 0..n |
| choice | 7.9 | 0..n |
| contact | 7.1.8 | 0..1 |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| deviation | 7.19.3 | 0..n |
| extension | 7.18 | 0..n |
| feature | 7.19.1 | 0..n |
| grouping | 7.11 | 0..n |
| identity | 7.17 | 0..n |
| import | 7.1.5 | 0..n |
| include | 7.1.6 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| namespace | 7.1.3 | 1 |
| notification | 7.15 | 0..n |
| organization | 7.1.7 | 0..1 |
| prefix | 7.1.4 | 1 |
| reference | 7.20.4 | 0..1 |
| revision | 7.1.9 | 0..n |
| rpc | 7.13 | 0..n |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |
| yang-version | 7.1.2 | 1 |

### 7.1.2. The yang-version Statement

The "yang-version" statement specifies which version of the YANG language was used in developing the module. The statement’s argument is a string. It MUST contain the value "1.1", which is the current YANG version.

A module or submodule that doesn’t contain the "yang-version" statement, or one that contains the value "1", is developed for YANG version 1, defined in [RFC6020].

Handling of the "yang-version" statement for versions other than "1.1" (the version defined here) is out of scope for this specification. Any document that defines a higher version will need to define the backward compatibility of such a higher version.

### 7.1.3. The namespace Statement

The "namespace" statement defines the XML namespace that all identifiers defined by the module are qualified by, with the exception of data node identifiers defined inside a grouping (see Section 7.12 for details). The argument to the "namespace" statement is the URI of the namespace.

See also Section 5.3.

### 7.1.4. The prefix Statement

The "prefix" statement is used to define the prefix associated with the module and its namespace. The "prefix" statement's argument is the prefix string that is used as a prefix to access a module. The prefix string MAY be used to refer to definitions contained in the module, e.g., "if:ifName". A prefix follows the same rules as an identifier (see Section 6.2).

When used inside the "module" statement, the "prefix" statement defines the prefix to be used when this module is imported. To improve readability of the NETCONF XML, a NETCONF client or server that generates XML or XPath that use prefixes SHOULD use the prefix defined by the module, unless there is a conflict.

When used inside the "import" statement, the "prefix" statement defines the prefix to be used when accessing definitions inside the imported module. When a reference to an identifier from the imported module is used, the prefix string for the imported module is used in combination with a colon (":") and the identifier, e.g., "if:ifIndex". To improve readability of YANG modules, the prefix defined by a module SHOULD be used when the module is imported, unless there is a conflict. If there is a conflict, i.e., two different modules that both have defined the same prefix are imported, at least one of them MUST be imported with a different prefix.

All prefixes, including the prefix for the module itself MUST be unique within the module or submodule.

### 7.1.5. The import Statement

The "import" statement makes definitions from one module available inside another module or submodule. The argument is the name of the module to import, and the statement is followed by a block of substatements that holds detailed import information. When a module is imported, the importing module may:

- o use any grouping and typedef defined at the top level in the imported module or its submodules.

- o use any extension, feature, and identity defined in the imported module or its submodules.

- o use any node in the imported module's schema tree in "must", "path", and "when" statements, or as the target node in "augment" and "deviation" statements.

The mandatory "prefix" substatement assigns a prefix for the imported module that is scoped to the importing module or submodule. Multiple "import" statements may be specified to import from different modules.

When the optional "revision-date" substatement is present, any typedef, grouping, extension, feature, and identity referenced by definitions in the local module are taken from the specified revision of the imported module. It is an error if the specified revision of the imported module does not exist. If no "revision-date" substatement is present, it is undefined from which revision of the module they are taken.

Multiple revisions of the same module MUST NOT be imported.

| substatement | section | cardinality |
|---|---|---|
| prefix | 7.1.4 | 1 |
| revision-date | 7.1.5.1 | 0..1 |

The import's Substatements

#### 7.1.5.1. The import's revision-date Statement

The import's "revision-date" statement is used to specify the exact version of the module to import. The "revision-date" statement MUST match the most recent "revision" statement in the imported module.

### 7.1.6. The include Statement

The "include" statement is used to make content from a submodule available to that submodule's parent module. The argument is an identifier that is the name of the submodule to include. Modules are only allowed to include submodules that belong to that module, as defined by the "belongs-to" statement (see Section 7.2.2).

Submodules are only allowed to include other submodules belonging to the same module.

When a module includes a submodule, it incorporates the contents of the submodule into the node hierarchy of the module.

For backwards compatibility with YANG version 1, a submodule is allowed to include another submodule belonging to the same module, but this is not necessary in YANG version 1.1.

When the optional "revision-date" substatement is present, the specified revision of the submodule is included in the module. It is an error if the specified revision of the submodule does not exist. If no "revision-date" substatement is present, it is undefined which revision of the submodule is included.

Multiple revisions of the same submodule MUST NOT be included.

| substatement | section | cardinality |
|---|---|---|
| revision-date | 7.1.5.1 | 0..1 |

The includes’s Substatements

### 7.1.7. The organization Statement

The "organization" statement defines the party responsible for this module. The argument is a string that is used to specify a textual description of the organization(s) under whose auspices this module was developed.

### 7.1.8. The contact Statement

The "contact" statement provides contact information for the module. The argument is a string that is used to specify contact information for the person or persons to whom technical queries concerning this module should be sent, such as their name, postal address, telephone number, and electronic mail address.

### 7.1.9. The revision Statement

The "revision" statement specifies the editorial revision history of the module, including the initial revision. A series of revision statements detail the changes in the module’s definition. The argument is a date string in the format "YYYY-MM-DD", followed by a block of substatements that holds detailed revision information. A

module SHOULD have at least one "revision" statement. For every published editorial change, a new one SHOULD be added in front of the revisions sequence, so that all revisions are in reverse chronological order.

#### 7.1.9.1. The revision's Substatement

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| reference | 7.20.4 | 0..1 |

### 7.1.10. Usage Example

```
module acme-system {
    yang-version 1.1;
    namespace "http://acme.example.com/system";
    prefix "acme";

    import ietf-yang-types {
        prefix "yang";
    }

    include acme-types;

    organization "ACME Inc.";
    contact
        "Joe L. User

         ACME, Inc.
         42 Anywhere Drive
         Nowhere, CA 95134
         USA

         Phone: +1 800 555 0100
         EMail: joe@acme.example.com";

    description
        "The module for entities implementing the ACME protocol.";

    revision "2007-06-09" {
        description "Initial revision.";
    }

    // definitions follow...
}
```

## 7.2. The submodule Statement

While the primary unit in YANG is a module, a YANG module can itself be constructed out of several submodules. Submodules allow a module designer to split a complex model into several pieces where all the submodules contribute to a single namespace, which is defined by the module that includes the submodules.

The "submodule" statement defines the submodule's name, and groups all statements that belong to the submodule together. The "submodule" statement's argument is the name of the submodule, followed by a block of substatements that hold detailed submodule information. The submodule name follows the rules for identifiers in Section 6.2.

Names of submodules published in RFC streams [RFC4844] MUST be assigned by IANA, see Section 15.

Private submodule names are assigned by the organization owning the submodule without a central registry. It is RECOMMENDED to choose submodule names that will have a low probability of colliding with standard or other enterprise modules and submodules, e.g., by using the enterprise or organization name as a prefix for the submodule name.

A submodule typically has the following layout:

```
submodule <module-name> {

    <yang-version statement>
```

```
        // module identification
        <belongs-to statement>

        // linkage statements
        <import statements>

        // meta information
        <organization statement>
        <contact statement>
        <description statement>
        <reference statement>

        // revision history
        <revision statements>

        // module definitions
        <other statements>
    }
```

### 7.2.1. The submodule's Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| augment | 7.16 | 0..n |
| belongs-to | 7.2.2 | 1 |
| choice | 7.9 | 0..n |
| contact | 7.1.8 | 0..1 |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| deviation | 7.19.3 | 0..n |
| extension | 7.18 | 0..n |
| feature | 7.19.1 | 0..n |
| grouping | 7.11 | 0..n |
| identity | 7.17 | 0..n |
| import | 7.1.5 | 0..n |
| include | 7.1.6 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| notification | 7.15 | 0..n |
| organization | 7.1.7 | 0..1 |
| reference | 7.20.4 | 0..1 |
| revision | 7.1.9 | 0..n |
| rpc | 7.13 | 0..n |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |
| yang-version | 7.1.2 | 1 |

### 7.2.2. The belongs-to Statement

The "belongs-to" statement specifies the module to which the submodule belongs. The argument is an identifier that is the name of the module.

A submodule MUST only be included by the module to which it belongs, or by another submodule that belongs to that module.

The mandatory "prefix" substatement assigns a prefix for the module to which the submodule belongs. All definitions in the module that the submodule belongs to and all its submodules can be accessed by using the prefix.

| substatement | section | cardinality |
|---|---|---|
| prefix | 7.1.4 | 1 |

The belongs-to's Substatements

### 7.2.3. Usage Example

```
submodule acme-types {
    yang-version 1.1;
    belongs-to "acme-system" {
        prefix "acme";
    }

    import ietf-yang-types {
        prefix "yang";
    }

    organization "ACME Inc.";
    contact
        "Joe L. User

         ACME, Inc.
         42 Anywhere Drive
         Nowhere, CA 95134
         USA

         Phone: +1 800 555 0100
         EMail: joe@acme.example.com";

    description
        "This submodule defines common ACME types.";

    revision "2007-06-09" {
        description "Initial revision.";
    }

    // definitions follows...
}
```

## 7.3. The typedef Statement

The "typedef" statement defines a new type that may be used locally in the module or submodule, and by other modules that import from it, according to the rules in Section 5.5. The new type is called the "derived type", and the type from which it was derived is called the

"base type". All derived types can be traced back to a YANG built-in type.

The "typedef" statement's argument is an identifier that is the name of the type to be defined, and MUST be followed by a block of substatements that holds detailed typedef information.

The name of the type MUST NOT be one of the YANG built-in types. If the typedef is defined at the top level of a YANG module or submodule, the name of the type to be defined MUST be unique within the module.

### 7.3.1. The typedef's Substatements

| substatement | section | cardinality |
|---|---|---|
| default | 7.3.4 | 0..1 |
| description | 7.20.3 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| type | 7.3.2 | 1 |
| units | 7.3.3 | 0..1 |

### 7.3.2. The typedef's type Statement

The "type" statement, which MUST be present, defines the base type from which this type is derived. See Section 7.4 for details.

### 7.3.3. The units Statement

The "units" statement, which is optional, takes as an argument a string that contains a textual definition of the units associated with the type.

### 7.3.4. The typedef's default Statement

The "default" statement takes as an argument a string that contains a default value for the new type.

The value of the "default" statement MUST be valid according to the type specified in the "type" statement.

If the base type has a default value, and the new derived type does not specify a new default value, the base type's default value is also the default value of the new derived type.

If the type’s default value is not valid according to the new restrictions specified in a derived type or leaf definition, the derived type or leaf definition MUST specify a new default value compatible with the restrictions.

### 7.3.5. Usage Example

```
typedef listen-ipv4-address {
    type inet:ipv4-address;
    default "0.0.0.0";
}
```

## 7.4. The type Statement

The "type" statement takes as an argument a string that is the name of a YANG built-in type (see Section 9) or a derived type (see Section 7.3), followed by an optional block of substatements that are used to put further restrictions on the type.

The restrictions that can be applied depend on the type being restricted. The restriction statements for all built-in types are described in the subsections of Section 9.

### 7.4.1. The type’s Substatements

| substatement | section | cardinality |
|---|---|---|
| base | 7.17.2 | 0..n |
| bit | 9.7.4 | 0..n |
| enum | 9.6.4 | 0..n |
| fraction-digits | 9.3.4 | 0..1 |
| length | 9.4.4 | 0..1 |
| path | 9.9.2 | 0..1 |
| pattern | 9.4.5 | 0..n |
| range | 9.2.4 | 0..1 |
| require-instance | 9.9.3 | 0..1 |
| type | 7.4 | 0..n |

## 7.5. The container Statement

The "container" statement is used to define an interior data node in the schema tree. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed container information.

A container node does not have a value, but it has a list of child nodes in the data tree. The child nodes are defined in the container’s substatements.

### 7.5.1. Containers with Presence

YANG supports two styles of containers, those that exist only for organizing the hierarchy of data nodes, and those whose presence in the configuration has an explicit meaning.

In the first style, the container has no meaning of its own, existing only to contain child nodes. This is the default style.

For example, the set of scrambling options for Synchronous Optical Network (SONET) interfaces may be placed inside a "scrambling" container to enhance the organization of the configuration hierarchy, and to keep these nodes together. The "scrambling" node itself has no meaning, so removing the node when it becomes empty relieves the user from performing this task.

In the second style, the presence of the container itself is configuration data, representing a single bit of configuration data. The container acts as both a configuration knob and a means of organizing related configuration. These containers are explicitly created and deleted.

YANG calls this style a "presence container" and it is indicated using the "presence" statement, which takes as its argument a text string indicating what the presence of the node means.

For example, an "ssh" container may turn on the ability to log into the device using ssh, but can also contain any ssh-related configuration knobs, such as connection rates or retry limits.

The "presence" statement (see Section 7.5.5) is used to give semantics to the existence of the container in the data tree.

### 7.5.2. The container’s Substatements

| substatement | section | cardinality |
|---|---|---|
| action | 7.14 | 0..n |
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| config | 7.20.1 | 0..1 |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| if-feature | 7.19.2 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| must | 7.5.3 | 0..n |
| presence | 7.5.5 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |
| when | 7.20.5 | 0..1 |

### 7.5.3. The must Statement

The "must" statement, which is optional, takes as an argument a string that contains an XPath expression (see Section 6.4). It is used to formally declare a constraint on valid data. The constraint is enforced according to the rules in Section 8.

When a datastore is validated, all "must" constraints are conceptually evaluated once for each data node in the accessible tree (see Section 6.4.1).

All such constraints MUST evaluate to true for the data to be valid.

The XPath expression is conceptually evaluated in the following context, in addition to the definition in Section 6.4.1:

- o The context node is the node in the accessible tree for which the "must" statement is defined.

The result of the XPath expression is converted to a boolean value using the standard XPath rules.

Note that since all leaf values in the data tree are conceptually stored in their canonical form (see Section 7.6 and Section 7.7), any XPath comparisons are done on the canonical value.

Also note that the XPath expression is conceptually evaluated. This means that an implementation does not have to use an XPath evaluator on the device. How the evaluation is done in practice is an implementation decision.

### 7.5.4. The must's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| error-app-tag | 7.5.4.2 | 0..1 |
| error-message | 7.5.4.1 | 0..1 |
| reference | 7.20.4 | 0..1 |

#### 7.5.4.1. The error-message Statement

The "error-message" statement, which is optional, takes a string as an argument. If the constraint evaluates to false, the string is passed as <error-message> in the <rpc-error>.

#### 7.5.4.2. The error-app-tag Statement

The "error-app-tag" statement, which is optional, takes a string as an argument. If the constraint evaluates to false, the string is passed as <error-app-tag> in the <rpc-error>.

#### 7.5.4.3. Usage Example of must and error-message

```
container interface {
    leaf ifType {
        type enumeration {
            enum ethernet;
            enum atm;
        }
    }
    leaf ifMTU {
        type uint32;
    }
    must "ifType != 'ethernet' or " +
         "(ifType = 'ethernet' and ifMTU = 1500)" {
        error-message "An ethernet MTU must be 1500";
    }
    must "ifType != 'atm' or " +
         "(ifType = 'atm' and ifMTU <= 17966 and ifMTU >= 64)" {
        error-message "An atm MTU must be  64 .. 17966";
    }
}
```

### 7.5.5. The presence Statement

The "presence" statement assigns a meaning to the presence of a container in the data tree. It takes as an argument a string that contains a textual description of what the node's presence means.

If a container has the "presence" statement, the container's existence in the data tree carries some meaning. Otherwise, the container is used to give some structure to the data, and it carries no meaning by itself.

See Section 7.5.1 for additional information.

### 7.5.6. The container's Child Node Statements

Within a container, the "container", "leaf", "list", "leaf-list", "uses", "choice", and "anyxml" statements can be used to define child nodes to the container.

### 7.5.7. XML Mapping Rules

A container node is encoded as an XML element. The element's local name is the container's identifier, and its namespace is the module's XML namespace (see Section 7.1.3).

The container's child nodes are encoded as subelements to the container element. If the container defines RPC input or output parameters, these subelements are encoded in the same order as they

are defined within the "container" statement. Otherwise, the subelements are encoded in any order.

A NETCONF server that replies to a <get> or <get-config> request MAY choose not to send a container element if the container node does not have the "presence" statement and no child nodes exist. Thus, a client that receives an <rpc-reply> for a <get> or <get-config> request, must be prepared to handle the case that a container node without a "presence" statement is not present in the XML.

### 7.5.8. NETCONF <edit-config> Operations

Containers can be created, deleted, replaced, and modified through <edit-config>, by using the "operation" attribute (see [RFC6241], Section 7.2) in the container’s XML element.

If a container does not have a "presence" statement and the last child node is deleted, the NETCONF server MAY delete the container.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the container node are:

If the operation is "merge" or "replace", the node is created if it does not exist.

If the operation is "create", the node is created if it does not exist. If the node already exists, a "data-exists" error is returned.

If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

### 7.5.9. Usage Example

Given the following container definition:

```
container system {
    description "Contains various system parameters";
    container services {
        description "Configure externally available services";
        container "ssh" {
            presence "Enables SSH";
            description "SSH service specific configuration";
            // more leafs, containers and stuff here...
        }
    }
}
```

A corresponding XML instance example:

```
<system>
  <services>
    <ssh/>
  </services>
</system>
```

Since the <ssh> element is present, ssh is enabled.

To delete a container with an <edit-config>:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <services>
          <ssh nc:operation="delete"/>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

## 7.6. The leaf Statement

The "leaf" statement is used to define a leaf node in the schema tree. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed leaf information.

A leaf node has a value, but no child nodes in the data tree. Conceptually, the value in the data tree is always in the canonical form (see Section 9.1).

A leaf node exists in zero or one instances in the data tree.

The "leaf" statement is used to define a scalar variable of a particular built-in or derived type.

### 7.6.1. The leaf's default value

The default value of a leaf is the value that the server uses if the leaf does not exist in the data tree. The usage of the default value depends on the leaf's closest ancestor node in the schema tree that is not a non-presence container:

- o If no such ancestor exists in the schema tree, the default value MUST be used.
- o Otherwise, if this ancestor is a case node, the default value MUST be used if any node from the case exists in the data tree, or if the case node is the choice's default case, and no nodes from any other case exist in the data tree.
- o Otherwise, the default value MUST be used if the ancestor node exists in the data tree.

In these cases, the default value is said to be in use.

When the default value is in use, the server MUST operationally behave as if the leaf was present in the data tree with the default value as its value.

If a leaf has a "default" statement, the leaf's default value is the value of the "default" statement. Otherwise, if the leaf's type has a default value, and the leaf is not mandatory, then the leaf's default value is the type's default value. In all other cases, the leaf does not have a default value.

### 7.6.2. The leaf's Substatements

| substatement | section | cardinality |
|---|---|---|
| config | 7.20.1 | 0..1 |
| default | 7.6.4 | 0..1 |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| mandatory | 7.6.5 | 0..1 |
| must | 7.5.3 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| type | 7.6.3 | 1 |
| units | 7.3.3 | 0..1 |
| when | 7.20.5 | 0..1 |

### 7.6.3. The leaf's type Statement

The "type" statement, which MUST be present, takes as an argument the name of an existing built-in or derived type. The optional substatements specify restrictions on this type. See Section 7.4 for details.

### 7.6.4. The leaf's default Statement

The "default" statement, which is optional, takes as an argument a string that contains a default value for the leaf.

The value of the "default" statement MUST be valid according to the type specified in the leaf's "type" statement.

The "default" statement MUST NOT be present on nodes where "mandatory" is true.

### 7.6.5. The leaf's mandatory Statement

The "mandatory" statement, which is optional, takes as an argument the string "true" or "false", and puts a constraint on valid data. If not specified, the default is "false".

If "mandatory" is "true", the behavior of the constraint depends on the type of the leaf's closest ancestor node in the schema tree that is not a non-presence container (see Section 7.5.1):

- o If no such ancestor exists in the schema tree, the leaf MUST exist.

- o Otherwise, if this ancestor is a case node, the leaf MUST exist if any node from the case exists in the data tree.

- o Otherwise, the leaf MUST exist if the ancestor node exists in the data tree.

This constraint is enforced according to the rules in Section 8.

### 7.6.6. XML Mapping Rules

A leaf node is encoded as an XML element. The element's local name is the leaf's identifier, and its namespace is the module's XML namespace (see Section 7.1.3).

The value of the leaf node is encoded to XML according to the type, and sent as character data in the element.

A NETCONF server that replies to a <get> or <get-config> request MAY choose not to send the leaf element if its value is the default value. Thus, a client that receives an <rpc-reply> for a <get> or <get-config> request, MUST be prepared to handle the case that a leaf node with a default value is not present in the XML. In this case, the value used by the server is known to be the default value.

See Section 7.6.8 for an example.

### 7.6.7. NETCONF <edit-config> Operations

When a NETCONF server processes an <edit-config> request, the elements of procedure for the leaf node are:

If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the value found in the XML RPC data.

If the operation is "create", the node is created if it does not exist. If the node already exists, a "data-exists" error is returned.

If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

### 7.6.8. Usage Example

Given the following "leaf" statement, placed in the previously defined "ssh" container (see Section 7.5.9):

```
leaf port {
    type inet:port-number;
    default 22;
    description "The port to which the SSH server listens"
}
```

A corresponding XML instance example:

```
<port>2022</port>
```

To set the value of a leaf with an <edit-config>:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <services>
          <ssh>
            <port>2022</port>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

### 7.7. The leaf-list Statement

Where the "leaf" statement is used to define a simple scalar variable of a particular type, the "leaf-list" statement is used to define an array of a particular type. The "leaf-list" statement takes one argument, which is an identifier, followed by a block of substatements that holds detailed leaf-list information.

The values in a leaf-list MUST be unique.

Conceptually, the values in the data tree are always in the canonical form (see Section 9.1).

#### 7.7.1. Ordering

YANG supports two styles for ordering the entries within lists and leaf-lists. In many lists, the order of list entries does not impact the implementation of the list’s configuration, and the device is free to sort the list entries in any reasonable order. The "description" string for the list may suggest an order to the device implementor. YANG calls this style of list "system ordered" and they are indicated with the statement "ordered-by system".

For example, a list of valid users would typically be sorted alphabetically, since the order in which the users appeared in the configuration would not impact the creation of those users’ accounts.

In the other style of lists, the order of list entries matters for the implementation of the list’s configuration and the user is

responsible for ordering the entries, while the device maintains that order. YANG calls this style of list "user ordered" and they are indicated with the statement "ordered-by user".

For example, the order in which firewall filters entries are applied to incoming traffic may affect how that traffic is filtered. The user would need to decide if the filter entry that discards all TCP traffic should be applied before or after the filter entry that allows all traffic from trusted interfaces. The choice of order would be crucial.

YANG provides a rich set of facilities within NETCONF's <edit-config> operation that allows the order of list entries in user-ordered lists to be controlled. List entries may be inserted or rearranged, positioned as the first or last entry in the list, or positioned before or after another specific entry.

The "ordered-by" statement is covered in Section 7.7.7.

### 7.7.2. The leaf-list's default values

The default values of a leaf-list are the values that the server uses if the leaf-list does not exist in the data tree. The usage of the default values depends on the leaf-list's closest ancestor node in the schema tree that is not a non-presence container:

- o If no such ancestor exists in the schema tree, the default values MUST be used.

- o Otherwise, if this ancestor is a case node, the default values MUST be used if any node from the case exists in the data tree, or if the case node is the choice's default case, and no nodes from any other case exist in the data tree.

- o Otherwise, the default values MUST be used if the ancestor node exists in the data tree.

In these cases, the default values are said to be in use.

When the default values are in use, the server MUST operationally behave as if the leaf-list was present in the data tree with the default values as its values.

If a leaf-list has one or more "default" statement, the leaf-list's default value are the values of the "default" statements, and if the leaf-list is user-ordered, the default values are used in the order of the "default" statements. Otherwise, if the leaf-list's type has a default value, and the leaf-list does not have a "min-elements"

statement with a value greater than or equal to one, then the leaf-list's default value is the type's default value. In all other cases, the leaf-list does not have any default values.

### 7.7.3. The leaf-list's Substatements

| substatement | section | cardinality |
|---|---|---|
| config | 7.20.1 | 0..1 |
| default | 7.7.4 | 0..n |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| max-elements | 7.7.6 | 0..1 |
| min-elements | 7.7.5 | 0..1 |
| must | 7.5.3 | 0..n |
| ordered-by | 7.7.7 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| type | 7.4 | 1 |
| units | 7.3.3 | 0..1 |
| when | 7.20.5 | 0..1 |

### 7.7.4. The leaf-list's default Statement

The "default" statement, which is optional, takes as an argument a string that contains a default value for the leaf-list.

The value of the "default" statement MUST be valid according to the type specified in the leaf-list's "type" statement.

The "default" statement MUST NOT be present on nodes where "min-elements" has a value greater than or equal to one.

### 7.7.5. The min-elements Statement

The "min-elements" statement, which is optional, takes as an argument a non-negative integer that puts a constraint on valid list entries. A valid leaf-list or list MUST have at least min-elements entries.

If no "min-elements" statement is present, it defaults to zero.

The behavior of the constraint depends on the type of the leaf-list's or list's closest ancestor node in the schema tree that is not a non-presence container (see Section 7.5.1):

- o If this ancestor is a case node, the constraint is enforced if any other node from the case exists.

- o Otherwise, it is enforced if the ancestor node exists.

The constraint is further enforced according to the rules in Section 8.

### 7.7.6. The max-elements Statement

The "max-elements" statement, which is optional, takes as an argument a positive integer or the string "unbounded", which puts a constraint on valid list entries. A valid leaf-list or list always has at most max-elements entries.

If no "max-elements" statement is present, it defaults to "unbounded".

The "max-elements" constraint is enforced according to the rules in Section 8.

### 7.7.7. The ordered-by Statement

The "ordered-by" statement defines whether the order of entries within a list are determined by the user or the system. The argument is one of the strings "system" or "user". If not present, order defaults to "system".

This statement is ignored if the list represents state data, RPC output parameters, or notification content.

See Section 7.7.1 for additional information.

#### 7.7.7.1. ordered-by system

The entries in the list are sorted according to an unspecified order. Thus, an implementation is free to sort the entries in the most appropriate order. An implementation SHOULD use the same order for the same data, regardless of how the data were created. Using a deterministic order will make comparisons possible using simple tools like "diff".

This is the default order.

#### 7.7.7.2. ordered-by user

The entries in the list are sorted according to an order defined by the user. This order is controlled by using special XML attributes in the <edit-config> request. See Section 7.7.9 for details.

### 7.7.8. XML Mapping Rules

A leaf-list node is encoded as a series of XML elements. Each element's local name is the leaf-list's identifier, and its namespace is the module's XML namespace (see Section 7.1.3).

The value of each leaf-list entry is encoded to XML according to the type, and sent as character data in the element.

The XML elements representing leaf-list entries MUST appear in the order specified by the user if the leaf-list is "ordered-by user"; otherwise, the order is implementation-dependent. The XML elements representing leaf-list entries MAY be interleaved with other sibling elements, unless the leaf-list defines RPC input or output parameters.

See Section 7.7.10 for an example.

### 7.7.9. NETCONF <edit-config> Operations

Leaf-list entries can be created and deleted, but not modified, through <edit-config>, by using the "operation" attribute in the leaf-list entry's XML element.

In an "ordered-by user" leaf-list, the attributes "insert" and "value" in the YANG XML namespace (Section 5.3.1) can be used to control where in the leaf-list the entry is inserted. These can be used during "create" operations to insert a new leaf-list entry, or during "merge" or "replace" operations to insert a new leaf-list entry or move an existing one.

The "insert" attribute can take the values "first", "last", "before", and "after". If the value is "before" or "after", the "value" attribute MUST also be used to specify an existing entry in the leaf-list.

If no "insert" attribute is present in the "create" operation, it defaults to "last".

If several entries in an "ordered-by user" leaf-list are modified in the same <edit-config> request, the entries are modified one at the time, in the order of the XML elements in the request.

In a <copy-config>, or an <edit-config> with a "replace" operation that covers the entire leaf-list, the leaf-list order is the same as the order of the XML elements in the request.

When a NETCONF server processes an <edit-config> request, the elements of procedure for a leaf-list node are:

If the operation is "merge" or "replace", the leaf-list entry is created if it does not exist.

If the operation is "create", the leaf-list entry is created if it does not exist. If the leaf-list entry already exists, a "data-exists" error is returned.

If the operation is "delete", the entry is deleted from the leaf-list if it exists. If the leaf-list entry does not exist, a "data-missing" error is returned.

### 7.7.10. Usage Example

```
leaf-list allow-user  {
    type string;
    description "A list of user name patterns to allow";
}
```

A corresponding XML instance example:

```
<allow-user>alice</allow-user>
<allow-user>bob</allow-user>
```

To create a new element in this list, using the default <edit-config> operation "merge":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <services>
          <ssh>
            <allow-user>eric</allow-user>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

Given the following ordered-by user leaf-list:

```
leaf-list cipher  {
    type string;
    ordered-by user;
    description "A list of ciphers";
}
```

The following would be used to insert a new cipher "blowfish-cbc" after "3des-cbc":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <services>
          <ssh>
            <cipher nc:operation="create"
                    yang:insert="after"
                    yang:value="3des-cbc">blowfish-cbc</cipher>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

## 7.8. The list Statement

The "list" statement is used to define an interior data node in the schema tree. A list node may exist in multiple instances in the data tree. Each such instance is known as a list entry. The "list" statement takes one argument, which is an identifier, followed by a block of substatements that holds detailed list information.

A list entry is uniquely identified by the values of the list’s keys, if defined.

### 7.8.1. The list’s Substatements

| substatement | section | cardinality |
|---|---|---|
| action | 7.14 | 0..n |
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| config | 7.20.1 | 0..1 |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| if-feature | 7.19.2 | 0..n |
| key | 7.8.2 | 0..1 |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| max-elements | 7.7.6 | 0..1 |
| min-elements | 7.7.5 | 0..1 |
| must | 7.5.3 | 0..n |
| ordered-by | 7.7.7 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |
| unique | 7.8.3 | 0..n |
| uses | 7.12 | 0..n |
| when | 7.20.5 | 0..1 |

### 7.8.2. The list’s key Statement

The "key" statement, which MUST be present if the list represents configuration, and MAY be present otherwise, takes as an argument a string that specifies a space-separated list of leaf identifiers of this list. A leaf identifier MUST NOT appear more than once in the key. Each such leaf identifier MUST refer to a child leaf of the list. The leafs can be defined directly in substatements to the list, or in groupings used in the list.

The combined values of all the leafs specified in the key are used to uniquely identify a list entry. All key leafs MUST be given values when a list entry is created. Thus, any default values in the key leafs or their types are ignored. It also implies that any mandatory statement in the key leafs are ignored.

A leaf that is part of the key can be of any built-in or derived type, except it MUST NOT be the built-in type "empty".

All key leafs in a list MUST have the same value for their "config" as the list itself.

The key string syntax is formally defined by the rule "key-arg" in Section 13.

### 7.8.3. The list’s unique Statement

The "unique" statement is used to put constraints on valid list entries. It takes as an argument a string that contains a space-separated list of schema node identifiers, which MUST be given in the descendant form (see the rule "descendant-schema-nodeid" in Section 13). Each such schema node identifier MUST refer to a leaf.

If one of the referenced leafs represents configuration data, then all of the referenced leafs MUST represent configuration data.

The "unique" constraint specifies that the combined values of all the leaf instances specified in the argument string, including leafs with default values, MUST be unique within all list entry instances in which all referenced leafs exist. The constraint is enforced according to the rules in Section 8.

The unique string syntax is formally defined by the rule "unique-arg" in Section 13.

#### 7.8.3.1. Usage Example

With the following list:

```
list server {
    key "name";
    unique "ip port";
    leaf name {
        type string;
    }
    leaf ip {
        type inet:ip-address;
    }
    leaf port {
        type inet:port-number;
    }
}
```

The following configuration is not valid:

```
<server>
  <name>smtp</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>

<server>
  <name>http</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>
```

The following configuration is valid, since the "http" and "ftp" list entries do not have a value for all referenced leafs, and are thus not taken into account when the "unique" constraint is enforced:

```
<server>
  <name>smtp</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>

<server>
  <name>http</name>
  <ip>192.0.2.1</ip>
</server>

<server>
  <name>ftp</name>
  <ip>192.0.2.1</ip>
</server>
```

### 7.8.4. The list’s Child Node Statements

Within a list, the "container", "leaf", "list", "leaf-list", "uses", "choice", and "anyxml" statements can be used to define child nodes to the list.

### 7.8.5. XML Mapping Rules

A list is encoded as a series of XML elements, one for each entry in the list. Each element’s local name is the list’s identifier, and its namespace is the module’s XML namespace (see Section 7.1.3).

The list’s key nodes are encoded as subelements to the list’s identifier element, in the same order as they are defined within the "key" statement.

The rest of the list’s child nodes are encoded as subelements to the list element, after the keys. If the list defines RPC input or output parameters, the subelements are encoded in the same order as they are defined within the "list" statement. Otherwise, the subelements are encoded in any order.

The XML elements representing list entries MUST appear in the order specified by the user if the list is "ordered-by user", otherwise the order is implementation-dependent. The XML elements representing list entries MAY be interleaved with other sibling elements, unless the list defines RPC input or output parameters.

### 7.8.6. NETCONF <edit-config> Operations

List entries can be created, deleted, replaced, and modified through <edit-config>, by using the "operation" attribute in the list’s XML element. In each case, the values of all keys are used to uniquely identify a list entry. If all keys are not specified for a list entry, a "missing-element" error is returned.

In an "ordered-by user" list, the attributes "insert" and "key" in the YANG XML namespace (Section 5.3.1) can be used to control where in the list the entry is inserted. These can be used during "create" operations to insert a new list entry, or during "merge" or "replace" operations to insert a new list entry or move an existing one.

The "insert" attribute can take the values "first", "last", "before", and "after". If the value is "before" or "after", the "key" attribute MUST also be used, to specify an existing element in the list. The value of the "key" attribute is the key predicates of the full instance identifier (see Section 9.13) for the list entry.

If no "insert" attribute is present in the "create" operation, it defaults to "last".

If several entries in an "ordered-by user" list are modified in the same <edit-config> request, the entries are modified one at the time, in the order of the XML elements in the request.

In a <copy-config>, or an <edit-config> with a "replace" operation that covers the entire list, the list entry order is the same as the order of the XML elements in the request.

When a NETCONF server processes an <edit-config> request, the elements of procedure for a list node are:

If the operation is "merge" or "replace", the list entry is created if it does not exist. If the list entry already exists

and the "insert" and "key" attributes are present, the list entry is moved according to the values of the "insert" and "key" attributes. If the list entry exists and the "insert" and "key" attributes are not present, the list entry is not moved.

If the operation is "create", the list entry is created if it does not exist. If the list entry already exists, a "data-exists" error is returned.

If the operation is "delete", the entry is deleted from the list if it exists. If the list entry does not exist, a "data-missing" error is returned.

### 7.8.7. Usage Example

Given the following list:

```
list user {
    key "name";
    config true;
    description "This is a list of users in the system.";

    leaf name {
        type string;
    }
    leaf type {
        type string;
    }
    leaf full-name {
        type string;
    }
}
```

A corresponding XML instance example:

```
<user>
  <name>fred</name>
  <type>admin</type>
  <full-name>Fred Flintstone</full-name>
</user>
```

To create a new user "barney":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <user nc:operation="create">
          <name>barney</name>
          <type>admin</type>
          <full-name>Barney Rubble</full-name>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

To change the type of "fred" to "superuser":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <user>
          <name>fred</name>
          <type>superuser</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

Given the following ordered-by user list:

```
     list user {
         description "This is a list of users in the system.";
         ordered-by user;
         config true;

         key "name";

         leaf name {
             type string;
         }
         leaf type {
             type string;
         }
         leaf full-name {
             type string;
         }
     }
```

The following would be used to insert a new user "barney" after the user "fred":

```
     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
          xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
          xmlns:yang="urn:ietf:params:xml:ns:yang:1">
       <edit-config>
         <target>
           <running/>
         </target>
         <config>
           <system xmlns="http://example.com/schema/config"
                xmlns:ex="http://example.com/schema/config">
             <user nc:operation="create"
                   yang:insert="after"
                   yang:key="[ex:name='fred']">
               <name>barney</name>
               <type>admin</type>
               <full-name>Barney Rubble</full-name>
             </user>
           </system>
         </config>
       </edit-config>
     </rpc>
```

The following would be used to move the user "barney" before the user "fred":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config"
           xmlns:ex="http://example.com/schema/config">
        <user nc:operation="merge"
              yang:insert="before"
              yang:key="[ex:name='fred']">
          <name>barney</name>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

## 7.9. The choice Statement

The "choice" statement defines a set of alternatives, only one of which may exist at any one time. The argument is an identifier, followed by a block of substatements that holds detailed choice information. The identifier is used to identify the choice node in the schema tree. A choice node does not exist in the data tree.

A choice consists of a number of branches, defined with the "case" substatement. Each branch contains a number of child nodes. The nodes from at most one of the choice's branches exist at the same time.

See Section 8.3.2 for additional information.

### 7.9.1. The choice's Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| case | 7.9.2 | 0..n |
| choice | 7.9 | 0..n |
| config | 7.20.1 | 0..1 |
| container | 7.5 | 0..n |
| default | 7.9.3 | 0..1 |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| mandatory | 7.9.4 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| when | 7.20.5 | 0..1 |

### 7.9.2. The choice’s case Statement

The "case" statement is used to define branches of the choice. It takes as an argument an identifier, followed by a block of substatements that holds detailed case information.

The identifier is used to identify the case node in the schema tree. A case node does not exist in the data tree.

Within a "case" statement, the "anyxml", "choice", "container", "leaf", "list", "leaf-list", and "uses" statements can be used to define child nodes to the case node. The identifiers of all these child nodes MUST be unique within all cases in a choice. For example, the following is illegal:

```
choice interface-type {     // This example is illegal YANG
    case a {
        leaf ethernet { ... }
    }
    case b {
        container ethernet { ...}
    }
}
```

As a shorthand, the "case" statement can be omitted if the branch contains a single "anyxml", "choice", "container", "leaf", "list", or "leaf-list" statement. In this case, the identifier of the case node

is the same as the identifier in the branch statement. The following example:

```
choice interface-type {
    container ethernet { ... }
}
```

is equivalent to:

```
choice interface-type {
    case ethernet {
        container ethernet { ... }
    }
}
```

The case identifier MUST be unique within a choice.

#### 7.9.2.1. The case's Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| uses | 7.12 | 0..n |
| when | 7.20.5 | 0..1 |

### 7.9.3. The choice's default Statement

The "default" statement indicates if a case should be considered as the default if no child nodes from any of the choice's cases exist. The argument is the identifier of the "case" statement. If the "default" statement is missing, there is no default case.

The "default" statement MUST NOT be present on choices where "mandatory" is true.

The default case is only important when considering the default values of nodes under the cases. The default values for nodes under

the default case are used if none of the nodes under any of the cases are present.

There MUST NOT be any mandatory nodes (Section 3.1) directly under the default case.

Default values for child nodes under a case are only used if one of the nodes under that case is present, or if that case is the default case. If none of the nodes under a case are present and the case is not the default case, the default values of the cases' child nodes are ignored.

In this example, the choice defaults to "interval", and the default value will be used if none of "daily", "time-of-day", or "manual" are present. If "daily" is present, the default value for "time-of-day" will be used.

```
container transfer {
    choice how {
        default interval;
        case interval {
            leaf interval {
                type uint16;
                default 30;
                units minutes;
            }
        }
        case daily {
            leaf daily {
                type empty;
            }
            leaf time-of-day {
                type string;
                units 24-hour-clock;
                default 1am;
            }
        }
        case manual {
            leaf manual {
                type empty;
            }
        }
    }
}
```

### 7.9.4. The choice’s mandatory Statement

The "mandatory" statement, which is optional, takes as an argument the string "true" or "false", and puts a constraint on valid data. If "mandatory" is "true", at least one node from exactly one of the choice’s case branches MUST exist.

If not specified, the default is "false".

The behavior of the constraint depends on the type of the choice’s closest ancestor node in the schema tree which is not a non-presence container (see Section 7.5.1):

- If this ancestor is a case node, the constraint is enforced if any other node from the case exists.

- Otherwise, it is enforced if the ancestor node exists.

The constraint is further enforced according to the rules in Section 8.

### 7.9.5. XML Mapping Rules

The choice and case nodes are not visible in XML.

The child nodes of the selected "case" statement MUST be encoded in the same order as they are defined in the "case" statement if they are part of an RPC input or output parameter definition. Otherwise, the subelements are encoded in any order.

### 7.9.6. NETCONF <edit-config> Operations

Since only one of the choice’s cases can be valid at any time, the creation of a node from one case implicitly deletes all nodes from all other cases. If an <edit-config> operation creates a node from a case, the NETCONF server will delete any existing nodes that are defined in other cases inside the choice.

### 7.9.7. Usage Example

Given the following choice:

```
container protocol {
    choice name {
        case a {
            leaf udp {
                type empty;
            }
        }
        case b {
            leaf tcp {
               type empty;
            }
        }
    }
}
```

A corresponding XML instance example:

```
<protocol>
  <tcp/>
</protocol>
```

To change the protocol from tcp to udp:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="http://example.com/schema/config">
        <protocol>
          <udp nc:operation="create"/>
        </protocol>
      </system>
    </config>
  </edit-config>
</rpc>
```

## 7.10. The anyxml Statement

The "anyxml" statement defines an interior node in the schema tree. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed anyxml information.

The "anyxml" statement is used to represent an unknown chunk of XML. No restrictions are placed on the XML. This can be useful, for

example, in RPC replies. An example is the <filter> parameter in the <get-config> operation.

An anyxml node cannot be augmented (see Section 7.16).

Since the use of anyxml limits the manipulation of the content, it is RECOMMENDED that the "anyxml" statement not be used to represent configuration data.

An anyxml node exists in zero or one instances in the data tree.

### 7.10.1. The anyxml's Substatements

| substatement | section | cardinality |
|---|---|---|
| config | 7.20.1 | 0..1 |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| mandatory | 7.6.5 | 0..1 |
| must | 7.5.3 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| when | 7.20.5 | 0..1 |

### 7.10.2. XML Mapping Rules

An anyxml node is encoded as an XML element. The element's local name is the anyxml's identifier, and its namespace is the module's XML namespace (see Section 7.1.3). The value of the anyxml node is encoded as XML content of this element.

Note that any prefixes used in the encoding are local to each instance encoding. This means that the same XML may be encoded differently by different implementations.

### 7.10.3. NETCONF <edit-config> Operations

An anyxml node is treated as an opaque chunk of data. This data can be modified in its entirety only.

Any "operation" attributes present on subelements of an anyxml node are ignored by the NETCONF server.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the anyxml node are:

If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the XML content of the anyxml node found in the XML RPC data.

If the operation is "create", the node is created if it does not exist, and its value is set to the XML content of the anyxml node found in the XML RPC data. If the node already exists, a "data-exists" error is returned.

If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

### 7.10.4. Usage Example

Given the following "anyxml" statement:

```
anyxml data;
```

The following are two valid encodings of the same anyxml value:

```
<data xmlns:if="http://example.com/ns/interface">
  <if:interface>
    <if:ifIndex>1</if:ifIndex>
  </if:interface>
</data>

<data>
  <interface xmlns="http://example.com/ns/interface">
    <ifIndex>1</ifIndex>
  </interface>
</data>
```

## 7.11. The grouping Statement

The "grouping" statement is used to define a reusable block of nodes, which may be used locally in the module or submodule, and by other modules that import from it, according to the rules in Section 5.5. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed grouping information.

The "grouping" statement is not a data definition statement and, as such, does not define any nodes in the schema tree.

A grouping is like a "structure" or a "record" in conventional programming languages.

Once a grouping is defined, it can be referenced in a "uses" statement (see Section 7.12). A grouping MUST NOT reference itself, neither directly nor indirectly through a chain of other groupings.

If the grouping is defined at the top level of a YANG module or submodule, the grouping's identifier MUST be unique within the module.

A grouping is more than just a mechanism for textual substitution, but defines a collection of nodes. Identifiers appearing inside the grouping are resolved relative to the scope in which the grouping is defined, not where it is used. Prefix mappings, type names, grouping names, and extension usage are evaluated in the hierarchy where the "grouping" statement appears. For extensions, this means that extensions are applied to the grouping node, not the uses node.

### 7.11.1. The grouping's Substatements

| substatement | section | cardinality |
|---|---|---|
| action | 7.14 | 0..n |
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |

### 7.11.2. Usage Example

```
import ietf-inet-types {
    prefix "inet";
}

grouping endpoint {
    description "A reusable endpoint group.";
    leaf ip {
        type inet:ip-address;
    }
    leaf port {
        type inet:port-number;
    }
}
```

## 7.12. The uses Statement

The "uses" statement is used to reference a "grouping" definition. It takes one argument, which is the name of the grouping.

The effect of a "uses" reference to a grouping is that the nodes defined by the grouping are copied into the current schema tree, and then updated according to the "refine" and "augment" statements.

The identifiers defined in the grouping are not bound to a namespace until the contents of the grouping are added to the schema tree via a "uses" statement that does not appear inside a "grouping" statement, at which point they are bound to the namespace of the current module.

### 7.12.1. The uses's Substatements

| substatement | section | cardinality |
|---|---|---|
| augment | 7.16 | 0..n |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| refine | 7.12.2 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| when | 7.20.5 | 0..1 |

### 7.12.2. The refine Statement

Some of the properties of each node in the grouping can be refined with the "refine" statement. The argument is a string that identifies a node in the grouping. This node is called the refine's target node. If a node in the grouping is not present as a target

node of a "refine" statement, it is not refined, and thus used exactly as it was defined in the grouping.

The argument string is a descendant schema node identifier (see Section 6.5).

The following refinements can be done:

- o A leaf or choice node may get a default value, or a new default value if it already had one.
- o Any node may get a specialized "description" string.
- o Any node may get a specialized "reference" string.
- o Any node may get a different "config" statement.
- o A leaf, anyxml, or choice node may get a different "mandatory" statement.
- o A container node may get a "presence" statement.
- o A leaf, leaf-list, list, container, or anyxml node may get additional "must" expressions.
- o A leaf-list or list node may get a different "min-elements" or "max-elements" statement.
- o A leaf, leaf-list, list, container, or anyxml node may get additional "if-feature" expressions.

#### 7.12.3. XML Mapping Rules

Each node in the grouping is encoded as if it was defined inline, even if it is imported from another module with another XML namespace.

#### 7.12.4. Usage Example

To use the "endpoint" grouping defined in Section 7.11.2 in a definition of an HTTP server in some other module, we can do:

```
import acme-system {
    prefix "acme";
}

container http-server {
    leaf name {
        type string;
    }
    uses acme:endpoint;
}
```

A corresponding XML instance example:

```
<http-server>
  <name>extern-web</name>
  <ip>192.0.2.1</ip>
  <port>80</port>
</http-server>
```

If port 80 should be the default for the HTTP server, default can be added:

```
container http-server {
    leaf name {
        type string;
    }
    uses acme:endpoint {
        refine port {
            default 80;
        }
    }
}
```

If we want to define a list of servers, and each server has the ip and port as keys, we can do:

```
list server {
    key "ip port";
    leaf name {
        type string;
    }
    uses acme:endpoint;
}
```

The following is an error:

```
container http-server {
    uses acme:endpoint;
    leaf ip {          // illegal - same identifier "ip" used twice
        type string;
    }
}
```

## 7.13. The rpc Statement

The "rpc" statement is used to define an RPC operation. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed rpc information. This argument is the name of the RPC, and is used as the element name directly under the <rpc> element, as designated by the substitution group "rpcOperation" in [RFC6241].

The "rpc" statement defines an rpc node in the schema tree. Under the rpc node, a schema node with the name "input", and a schema node with the name "output" are also defined. The nodes "input" and "output" are defined in the module's namespace.

### 7.13.1. The rpc's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| if-feature | 7.19.2 | 0..n |
| input | 7.13.2 | 0..1 |
| output | 7.13.3 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |

### 7.13.2. The input Statement

The "input" statement, which is optional, is used to define input parameters to the operation. It does not take an argument. The substatements to "input" define nodes under the operation's input node.

If a leaf in the input tree has a "mandatory" statement with the value "true", the leaf MUST be present in a NETCONF RPC invocation. Otherwise, the server MUST return a "missing-element" error.

If a leaf in the input tree has a default value, the NETCONF server MUST use this value in the same cases as described in Section 7.6.1. In these cases, the server MUST operationally behave as if the leaf was present in the NETCONF RPC invocation with the default value as its value.

If a leaf-list in the input tree has one or more default values, the NETCONF server MUST use these values in the same cases as described in Section 7.7.2. In these cases, the server MUST operationally behave as if the leaf-list was present in the NETCONF RPC invocation with the default values as its values.

If a "config" statement is present for any node in the input tree, the "config" statement is ignored.

If any node has a "when" statement that would evaluate to false, then this node MUST NOT be present in the input tree.

#### 7.13.2.1. The input’s Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| grouping | 7.11 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| must | 7.5.3 | 0..n |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |

### 7.13.3. The output Statement

The "output" statement, which is optional, is used to define output parameters to the RPC operation. It does not take an argument. The substatements to "output" define nodes under the operation’s output node.

If a leaf in the output tree has a "mandatory" statement with the value "true", the leaf MUST be present in a NETCONF RPC reply.

If a leaf in the output tree has a default value, the NETCONF client MUST use this value in the same cases as described in Section 7.6.1. In these cases, the client MUST operationally behave as if the leaf

was present in the NETCONF RPC reply with the default value as its value.

If a leaf-list in the output tree has one or more default values, the NETCONF client MUST use these values in the same cases as described in Section 7.7.2. In these cases, the client MUST operationally behave as if the leaf-list was present in the NETCONF RPC reply with the default values as its values.

If a "config" statement is present for any node in the output tree, the "config" statement is ignored.

If any node has a "when" statement that would evaluate to false, then this node MUST NOT be present in the output tree.

#### 7.13.3.1. The output's Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| grouping | 7.11 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| must | 7.5.3 | 0..n |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |

### 7.13.4. XML Mapping Rules

An rpc node is encoded as a child XML element to the <rpc> element defined in [RFC6241]. The element's local name is the rpc's identifier, and its namespace is the module's XML namespace (see Section 7.1.3).

Input parameters are encoded as child XML elements to the rpc node's XML element, in the same order as they are defined within the "input" statement.

If the RPC operation invocation succeeded, and no output parameters are returned, the <rpc-reply> contains a single <ok/> element defined in [RFC6241]. If output parameters are returned, they are encoded as child elements to the <rpc-reply> element defined in [RFC6241], in the same order as they are defined within the "output" statement.

### 7.13.5. Usage Example

The following example defines an RPC operation:

```
module rock {
    yang-version 1.1;
    namespace "http://example.net/rock";
    prefix "rock";

    rpc rock-the-house {
        input {
            leaf zip-code {
                type string;
            }
        }
    }
}
```

A corresponding XML instance example of the complete rpc and rpc-reply:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="http://example.net/rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

## 7.14. The action Statement

The "action" statement is used to define an operation connected to a specific container or list data node. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed action information. The argument is the name of the action.

The "action" statement defines an action node in the schema tree. Under the action node, a schema node with the name "input", and a schema node with the name "output" are also defined. The nodes "input" and "output" are defined in the module’s namespace.

An action MUST NOT be defined within an rpc, another action or a notification, i.e., an action node MUST NOT have an rpc, action, or a notification node as one of its ancestors in the schema tree. For

example, this means that it is an error if a grouping that contains an action is used in a notification definition.

The difference between an action and an rpc is that an action is tied to a node in the data tree, whereas an rpc is not. When an action is invoked, the node in the data tree is specified along with the name of the action and the input parameters.

### 7.14.1. The action's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| if-feature | 7.19.2 | 0..n |
| input | 7.13.2 | 0..1 |
| output | 7.13.3 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |

### 7.14.2. XML Mapping Rules

When an action is invoked, an element with the local name "action" in the namespace "urn:ietf:params:xml:ns:yang:1" (see Section 5.3.1) is encoded as a child XML element to the <rpc> element defined in [RFC6241], as designated by the substitution group "rpcOperation" in [RFC6241].

The "action" element contains an hierarchy of nodes that identifies the node in the data tree. It MUST contain all containers and list nodes from the top level down to the list or container containing the action. For lists, all key leafs MUST also be included. The last container or list contains an XML element that carries the name of the defined action. Within this element the input parameters are encoded as child XML elements, in the same order as they are defined within the "input" statement.

Only one action can be invoked in one <rpc>. If more than one actions are present in the <rpc>, the server MUST reply with an "bad-element" error-tag in the <rpc-error>.

If the action operation invocation succeeded, and no output parameters are returned, the <rpc-reply> contains a single <ok/> element defined in [RFC6241]. If output parameters are returned, they are encoded as child elements to the <rpc-reply> element defined

in [RFC6241], in the same order as they are defined within the "output" statement.

### 7.14.3. Usage Example

The following example defines an action to reset one server at a server farm:

```
  module server-farm {
      yang-version 1.1;
      namespace "http://example.net/server-farm";
      prefix "sfarm";

       import ietf-yang-types {
           prefix "yang";
       }

       list server {
          key name;
          leaf name {
              type string;
          }
          action reset {
              input {
                  leaf reset-at {
                      type yang:date-and-time;
                      mandatory true;
                  }
              }
              output {
                  leaf reset-finished-at {
                     type yang:date-and-time;
                     mandatory true;
                  }
              }
          }
      }
   }
```

A corresponding XML instance example of the complete rpc and rpc-reply:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <server xmlns="http://example.net/server-farm">
      <name>apache-1</name>
      <reset>
        <reset-at>2014-07-29T13:42:00Z</reset-at>
      </reset>
    </server>
  </action>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <reset-finished-at xmlns="http://example.net/server-farm">
    2014-07-29T13:42:12Z
  </reset-at>
</rpc-reply>
```

## 7.15. The notification Statement

The "notification" statement is used to define a NETCONF notification. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed notification information. The "notification" statement defines a notification node in the schema tree.

If a leaf in the notification tree has a "mandatory" statement with the value "true", the leaf MUST be present in a NETCONF notification.

If a leaf in the notification tree has a default value, the NETCONF client MUST use this value in the same cases as described in Section 7.6.1. In these cases, the client MUST operationally behave as if the leaf was present in the NETCONF notification with the default value as its value.

If a leaf-list in the notification tree has one or more default values, the NETCONF client MUST use these values in the same cases as described in Section 7.7.2. In these cases, the client MUST operationally behave as if the leaf-list was present in the NETCONF notification with the default values as its values.

If a "config" statement is present for any node in the notification tree, the "config" statement is ignored.

### 7.15.1. The notification's Substatements

| substatement | section | cardinality |
|---|---|---|
| anyxml | 7.10 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| grouping | 7.11 | 0..n |
| if-feature | 7.19.2 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| must | 7.5.3 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| typedef | 7.3 | 0..n |
| uses | 7.12 | 0..n |

### 7.15.2. XML Mapping Rules

A notification node is encoded as a child XML element to the <notification> element defined in NETCONF Event Notifications [RFC5277]. The element's local name is the notification's identifier, and its namespace is the module's XML namespace (see Section 7.1.3).

### 7.15.3. Usage Example

The following example defines a notification:

```
module event {
    yang-version 1.1;
    namespace "http://example.com/event";
    prefix "ev";

    notification event {
        leaf event-class {
            type string;
        }
        anyxml reporting-entity;
        leaf severity {
            type string;
        }
    }
}
```

A corresponding XML instance example of the complete notification:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-07-08T00:01:00Z</eventTime>
  <event xmlns="http://example.com/event">
    <event-class>fault</event-class>
    <reporting-entity>
      <card>Ethernet0</card>
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

## 7.16. The augment Statement

The "augment" statement allows a module or submodule to add to the schema tree defined in an external module, or the current module and its submodules, and to add to the nodes from a grouping in a "uses" statement. The argument is a string that identifies a node in the schema tree. This node is called the augment’s target node. The target node MUST be either a container, list, choice, case, input, output, or notification node. It is augmented with the nodes defined in the substatements that follow the "augment" statement.

The argument string is a schema node identifier (see Section 6.5). If the "augment" statement is on the top level in a module or submodule, the absolute form (defined by the rule "absolute-schema-nodeid" in Section 13) of a schema node identifier MUST be used. If the "augment" statement is a substatement to the "uses" statement, the descendant form (defined by the rule "descendant-schema-nodeid" in Section 13) MUST be used.

If the target node is a container, list, case, input, output, or notification node, the "container", "leaf", "list", "leaf-list", "uses", and "choice" statements can be used within the "augment" statement.

If the target node is a choice node, the "case" statement, or a case shorthand statement (see Section 7.9.2) can be used within the "augment" statement.

If the target node is in another module, then nodes added by the augmentation MUST NOT be mandatory nodes (see Section 3.1).

The "augment" statement MUST NOT add multiple nodes with the same name from the same module to the target node.

### 7.16.1. The augment's Substatements

| substatement | section | cardinality |
|---|---|---|
| action | 7.14 | 0..n |
| anyxml | 7.10 | 0..n |
| case | 7.9.2 | 0..n |
| choice | 7.9 | 0..n |
| container | 7.5 | 0..n |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| leaf | 7.6 | 0..n |
| leaf-list | 7.7 | 0..n |
| list | 7.8 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| uses | 7.12 | 0..n |
| when | 7.20.5 | 0..1 |

### 7.16.2. XML Mapping Rules

All data nodes defined in the "augment" statement are defined as XML elements in the XML namespace of the module where the "augment" is specified.

When a node is augmented, the augmenting child nodes are encoded as subelements to the augmented node, in any order.

### 7.16.3. Usage Example

In namespace http://example.com/schema/interfaces, we have:

```
container interfaces {
    list ifEntry {
        key "ifIndex";

        leaf ifIndex {
            type uint32;
        }
        leaf ifDescr {
            type string;
        }
        leaf ifType {
            type iana:IfType;
        }
        leaf ifMtu {
            type int32;
        }
    }
}
```

Then, in namespace http://example.com/schema/ds0, we have:

```
import interface-module {
    prefix "if";
}
augment "/if:interfaces/if:ifEntry" {
    when "if:ifType='ds0'";
    leaf ds0ChannelNumber {
        type ChannelNumber;
    }
}
```

A corresponding XML instance example:

```
<interfaces xmlns="http://example.com/schema/interfaces"
            xmlns:ds0="http://example.com/schema/ds0">
  <ifEntry>
    <ifIndex>1</ifIndex>
    <ifDescr>Flintstone Inc Ethernet A562</ifDescr>
    <ifType>ethernetCsmacd</ifType>
    <ifMtu>1500</ifMtu>
  </ifEntry>
  <ifEntry>
    <ifIndex>2</ifIndex>
    <ifDescr>Flintstone Inc DS0</ifDescr>
    <ifType>ds0</ifType>
    <ds0:ds0ChannelNumber>1</ds0:ds0ChannelNumber>
  </ifEntry>
</interfaces>
```

As another example, suppose we have the choice defined in Section 7.9.7. The following construct can be used to extend the protocol definition:

```
augment /ex:system/ex:protocol/ex:name {
    case c {
        leaf smtp {
            type empty;
        }
    }
}
```

A corresponding XML instance example:

```
<ex:system>
  <ex:protocol>
    <ex:tcp/>
  </ex:protocol>
</ex:system>
```

or

```
<ex:system>
  <ex:protocol>
    <other:smtp/>
  </ex:protocol>
</ex:system>
```

## 7.17. The identity Statement

The "identity" statement is used to define a new globally unique, abstract, and untyped identity. Its only purpose is to denote its name, semantics, and existence. An identity can either be defined from scratch or derived from one or more base identities. The identity’s argument is an identifier that is the name of the identity. It is followed by a block of substatements that holds detailed identity information.

The built-in datatype "identityref" (see Section 9.10) can be used to reference identities within a data model.

### 7.17.1. The identity’s Substatements

| substatement | section | cardinality |
|---|---|---|
| base | 7.17.2 | 0..n |
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |

### 7.17.2. The base Statement

The "base" statement, which is optional, takes as an argument a string that is the name of an existing identity, from which the new identity is derived. If no "base" statement is present, the identity is defined from scratch. If multiple "base" statements are present, the identity is derived from all of them.

If a prefix is present on the base name, it refers to an identity defined in the module that was imported with that prefix, or the local module if the prefix matches the local module’s prefix. Otherwise, an identity with the matching name MUST be defined in the current module or an included submodule.

An identity MUST NOT reference itself, neither directly nor indirectly through a chain of other identities.

The derivation of identities has the following properties:

- It is irreflexive, which means that an identity is not derived from itself.

- It is transitive, which means that if identity B is derived from A and C is derived from B, then C is also derived from A.

### 7.17.3. Usage Example

```
module crypto-base {
    yang-version 1.1;
    namespace "http://example.com/crypto-base";
    prefix "crypto";

    identity crypto-alg {
        description
           "Base identity from which all crypto algorithms
            are derived.";
    }
}

module des {
    yang-version 1.1;
    namespace "http://example.com/des";
    prefix "des";

    import "crypto-base" {
        prefix "crypto";
    }

    identity des {
        base "crypto:crypto-alg";
        description "DES crypto algorithm";
    }

    identity des3 {
        base "crypto:crypto-alg";
        description "Triple DES crypto algorithm";
    }
}
```

## 7.18. The extension Statement

The "extension" statement allows the definition of new statements within the YANG language. This new statement definition can be imported and used by other modules.

The statement’s argument is an identifier that is the new keyword for the extension and must be followed by a block of substatements that holds detailed extension information. The purpose of the "extension" statement is to define a keyword, so that it can be imported and used by other modules.

The extension can be used like a normal YANG statement, with the statement name followed by an argument if one is defined by the "extension" statement, and an optional block of substatements. The statement’s name is created by combining the prefix of the module in

which the extension was defined, a colon (":"), and the extension's keyword, with no interleaving whitespace. The substatements of an extension are defined by the "extension" statement, using some mechanism outside the scope of this specification. Syntactically, the substatements MUST be YANG statements, or also extensions defined using "extension" statements. YANG statements in extensions MUST follow the syntactical rules in Section 13.

### 7.18.1. The extension's Substatements

| substatement | section | cardinality |
|---|---|---|
| argument | 7.18.2 | 0..1 |
| description | 7.20.3 | 0..1 |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |

### 7.18.2. The argument Statement

The "argument" statement, which is optional, takes as an argument a string that is the name of the argument to the keyword. If no argument statement is present, the keyword expects no argument when it is used.

The argument's name is used in the YIN mapping, where it is used as an XML attribute or element name, depending on the argument's "yin-element" statement.

#### 7.18.2.1. The argument's Substatements

| substatement | section | cardinality |
|---|---|---|
| yin-element | 7.18.2.2 | 0..1 |

#### 7.18.2.2. The yin-element Statement

The "yin-element" statement, which is optional, takes as an argument the string "true" or "false". This statement indicates if the argument is mapped to an XML element in YIN or to an XML attribute (see Section 12).

If no "yin-element" statement is present, it defaults to "false".

### 7.18.3. Usage Example

To define an extension:

```
module my-extensions {
  ...

  extension c-define {
    description
      "Takes as argument a name string.
      Makes the code generator use the given name in the
      #define.";
    argument "name";
  }
}
```

To use the extension:

```
module my-interfaces {
  ...
  import my-extensions {
    prefix "myext";
  }
  ...

  container interfaces {
    ...
    myext:c-define "MY_INTERFACES";
  }
}
```

## 7.19. Conformance-Related Statements

This section defines statements related to conformance, as described in Section 5.6.

### 7.19.1. The feature Statement

The "feature" statement is used to define a mechanism by which portions of the schema are marked as conditional. A feature name is defined that can later be referenced using the "if-feature" statement (see Section 7.19.2). Schema nodes tagged with an "if-feature" statement are ignored by the device unless the device supports the given feature expression. This allows portions of the YANG module to be conditional based on conditions on the device. The model can represent the abilities of the device within the model, giving a richer model that allows for differing device abilities and roles.

The argument to the "feature" statement is the name of the new feature, and follows the rules for identifiers in Section 6.2. This name is used by the "if-feature" statement to tie the schema nodes to the feature.

In this example, a feature called "local-storage" represents the ability for a device to store syslog messages on local storage of some sort. This feature is used to make the "local-storage-limit" leaf conditional on the presence of some sort of local storage. If the device does not report that it supports this feature, the "local-storage-limit" node is not supported.

```
module syslog {
    ...
    feature local-storage {
        description
            "This feature means the device supports local
             storage (memory, flash or disk) that can be used to
             store syslog messages.";
    }

    container syslog {
        leaf local-storage-limit {
            if-feature local-storage;
            type uint64;
            units "kilobyte";
            config false;
            description
                "The amount of local storage that can be
                 used to hold syslog messages.";
        }
    }
}
```

The "if-feature" statement can be used in many places within the YANG syntax. Definitions tagged with "if-feature" are ignored when the device does not support that feature.

A feature MUST NOT reference itself, neither directly nor indirectly through a chain of other features.

In order for a device to implement a feature that is dependent on any other features (i.e., the feature has one or more "if-feature" substatements), the device MUST also implement all the dependant features.

#### 7.19.1.1. The feature’s Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| status | 7.20.2 | 0..1 |
| reference | 7.20.4 | 0..1 |

### 7.19.2. The if-feature Statement

The "if-feature" statement makes its parent statement conditional. The argument is a boolean expression over feature names. In this expression, a feature name evaluates to "true" if and only if the feature is implemented by the server. The parent statement is implemented by servers where the boolean expression evaluates to "true".

The if-feature boolean expression syntax is formally defined by the rule "if-feature-expr" in Section 13. When this boolean expression is evaluated, the operator order of precedence is (highest precedence first): "not", "and", "or".

If a prefix is present on a feature name in the boolean expression, the prefixed name refers to a feature defined in the module that was imported with that prefix, or the local module if the prefix matches the local module’s prefix. Otherwise, a feature with the matching name MUST be defined in the current module or an included submodule.

A leaf that is a list key MUST NOT have any "if-feature" statements, unless the conditions specified in the "if-feature" statements are the same as the "if-feature" conditions in effect on the leaf’s parent node.

#### 7.19.2.1. Usage Example

In this example, the container "target" is implemented if any of the features "outbound-tls" or "outbound-ssh" is implemented by the server.

```
container target {
    if-feature "outbound-tls or outbound-ssh";
    ...
}
```

### 7.19.3. The deviation Statement

The "deviation" statement defines a hierarchy of a module that the device does not implement faithfully. The argument is a string that identifies the node in the schema tree where a deviation from the module occurs. This node is called the deviation's target node. The contents of the "deviation" statement give details about the deviation.

The argument string is an absolute schema node identifier (see Section 6.5).

Deviations define the way a device or class of devices deviate from a standard. This means that deviations MUST never be part of a published standard, since they are the mechanism for learning how implementations vary from the standards.

Device deviations are strongly discouraged and MUST only be used as a last resort. Telling the application how a device fails to follow a standard is no substitute for implementing the standard correctly. A device that deviates from a module is not fully compliant with the module.

However, in some cases, a particular device may not have the hardware or software ability to support parts of a standard module. When this occurs, the device makes a choice either to treat attempts to configure unsupported parts of the module as an error that is reported back to the unsuspecting application or ignore those incoming requests. Neither choice is acceptable.

Instead, YANG allows devices to document portions of a base module that are not supported or supported but with different syntax, by using the "deviation" statement.

#### 7.19.3.1. The deviation's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| deviate | 7.19.3.2 | 1..n |
| reference | 7.20.4 | 0..1 |

#### 7.19.3.2. The deviate Statement

The "deviate" statement defines how the device's implementation of the target node deviates from its original definition. The argument is one of the strings "not-supported", "add", "replace", or "delete".

The argument "not-supported" indicates that the target node is not implemented by this device.

The argument "add" adds properties to the target node. The properties to add are identified by substatements to the "deviate" statement. If a property can only appear once, the property MUST NOT exist in the target node.

The argument "replace" replaces properties of the target node. The properties to replace are identified by substatements to the "deviate" statement. The properties to replace MUST exist in the target node.

The argument "delete" deletes properties from the target node. The properties to delete are identified by substatements to the "delete" statement. The substatement's keyword MUST match a corresponding keyword in the target node, and the argument's string MUST be equal to the corresponding keyword's argument string in the target node.

| substatement | section | cardinality |
|---|---|---|
| config | 7.20.1 | 0..1 |
| default | 7.6.4 7.7.4 | 0..n |
| mandatory | 7.6.5 | 0..1 |
| max-elements | 7.7.6 | 0..1 |
| min-elements | 7.7.5 | 0..1 |
| must | 7.5.3 | 0..n |
| type | 7.4 | 0..1 |
| unique | 7.8.3 | 0..n |
| units | 7.3.3 | 0..1 |

The deviate's Substatements

#### 7.19.3.3. Usage Example

In this example, the device is informing client applications that it does not support the "daytime" service in the style of RFC 867.

```
deviation /base:system/base:daytime {
    deviate not-supported;
}
```

The following example sets a device-specific default value to a leaf that does not have a default value defined:

```
deviation /base:system/base:user/base:type {
    deviate add {
        default "admin"; // new users are 'admin' by default
    }
}
```

In this example, the device limits the number of name servers to 3:

```
deviation /base:system/base:name-server {
    deviate replace {
        max-elements 3;
    }
}
```

If the original definition is:

```
container system {
    must "daytime or time";
    ...
}
```

a device might remove this must constraint by doing:

```
deviation "/base:system" {
    deviate delete {
        must "daytime or time";
    }
}
```

## 7.20. Common Statements

This section defines substatements common to several other statements.

### 7.20.1. The config Statement

The "config" statement takes as an argument the string "true" or "false". If "config" is "true", the definition represents configuration. Data nodes representing configuration will be part of the reply to a <get-config> request, and can be sent in a <copy-config> or <edit-config> request.

If "config" is "false", the definition represents state data. Data nodes representing state data will be part of the reply to a <get>, but not to a <get-config> request, and cannot be sent in a <copy-config> or <edit-config> request.

If "config" is not specified, the default is the same as the parent schema node’s "config" value. If the parent node is a "case" node, the value is the same as the "case" node’s parent "choice" node.

If the top node does not specify a "config" statement, the default is "true".

If a node has "config" set to "false", no node underneath it can have "config" set to "true".

### 7.20.2. The status Statement

The "status" statement takes as an argument one of the strings "current", "deprecated", or "obsolete".

- "current" means that the definition is current and valid.

- "deprecated" indicates an obsolete definition, but it permits new/continued implementation in order to foster interoperability with older/existing implementations.

- "obsolete" means the definition is obsolete and SHOULD NOT be implemented and/or can be removed from implementations.

If no status is specified, the default is "current".

If a definition is "current", it MUST NOT reference a "deprecated" or "obsolete" definition within the same module.

If a definition is "deprecated", it MUST NOT reference an "obsolete" definition within the same module.

For example, the following is illegal:

```
typedef my-type {
  status deprecated;
  type int32;
}

leaf my-leaf {
  status current;
  type my-type; // illegal, since my-type is deprecated
}
```

### 7.20.3. The description Statement

The "description" statement takes as an argument a string that contains a human-readable textual description of this definition. The text is provided in a language (or languages) chosen by the module developer; for the sake of interoperability, it is RECOMMENDED to choose a language that is widely understood among the community of network administrators who will use the module.

### 7.20.4. The reference Statement

The "reference" statement takes as an argument a string that is used to specify a textual cross-reference to an external document, either another module that defines related management information, or a document that provides additional information relevant to this definition.

For example, a typedef for a "uri" data type could look like:

```
typedef uri {
  type string;
  reference
    "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
  ...
}
```

### 7.20.5. The when Statement

The "when" statement makes its parent data definition statement conditional. The node defined by the parent data definition statement is only valid when the condition specified by the "when" statement is satisfied. The statement’s argument is an XPath expression (see Section 6.4), which is used to formally specify this condition. If the XPath expression conceptually evaluates to "true" for a particular instance, then the node defined by the parent data definition statement is valid; otherwise, it is not.

A leaf that is a list key MUST NOT have a "when" statement, unless the condition specified in the "when" statement is the same as the "when" condition in effect on the leaf’s parent node.

See Section 8.3.2 for additional information.

The XPath expression is conceptually evaluated in the following context, in addition to the definition in Section 6.4.1:

- If the "when" statement is a child of an "augment" statement, then the context node is the augment’s target node in the data tree, if

the target node is a data node. Otherwise, the context node is the closest ancestor node to the target node that is also a data node.

- o If the "when" statement is a child of a "uses", "choice", or "case" statement, then the context node is the closest ancestor node to the "uses", "choice", or "case" node that is also a data node.

- o If the "when" statement is a child of any other data definition statement, the context node is the node in the accessible tree for which the "when" statement is defined.

The result of the XPath expression is converted to a boolean value using the standard XPath rules.

If the XPath expression references any node that also has associated "when" statements, these "when" expressions MUST be evaluated first. There MUST NOT be any circular dependencies in these "when" expressions.

Note that the XPath expression is conceptually evaluated. This means that an implementation does not have to use an XPath evaluator on the device. The "when" statement can very well be implemented with specially written code.

# 8. Constraints

## 8.1. Constraints on Data

Several YANG statements define constraints on valid data. These constraints are enforced in different ways, depending on what type of data the statement defines.

- o If the constraint is defined on configuration data, it MUST be true in a valid configuration data tree.

- o If the constraint is defined on state data, it MUST be true in a reply to a <get> operation without a filter.

- o If the constraint is defined on notification content, it MUST be true in any notification instance.

- o If the constraint is defined on RPC input parameters, it MUST be true in an invocation of the RPC operation.

- o If the constraint is defined on RPC output parameters, it MUST be true in the RPC reply.

## 8.2. Hierarchy of Constraints

Conditions on parent nodes affect constraints on child nodes as a natural consequence of the hierarchy of nodes. "must", "mandatory", "min-elements", and "max-elements" constraints are not enforced if the parent node has a "when" or "if-feature" property that is not satisfied on the current device.

In this example, the "mandatory" constraint on the "longitude" leaf is not enforced on devices that lack the "has-gps" feature:

```
container location {
    if-feature has-gps;
    leaf longitude {
        mandatory true;
        ...
    }
}
```

## 8.3. Constraint Enforcement Model

For configuration data, there are three windows when constraints MUST be enforced:

- o during parsing of RPC payloads
- o during processing of NETCONF operations
- o during validation

Each of these scenarios is considered in the following sections.

### 8.3.1. Payload Parsing

When content arrives in RPC payloads, it MUST be well-formed XML, following the hierarchy and content rules defined by the set of models the device implements.

- o If a leaf data value does not match the type constraints for the leaf, including those defined in the type’s "range", "length", and "pattern" properties, the server MUST reply with an "invalid-value" error-tag in the rpc-error, and with the error-app-tag and error-message associated with the constraint, if any exist.
- o If all keys of a list entry are not present, the server MUST reply with a "missing-element" error-tag in the rpc-error.

- o If data for more than one case branch of a choice is present, the server MUST reply with a "bad-element" in the rpc-error.

- o If data for a node tagged with "if-feature" is present, and the if-feature expression evaluates to "false" on the device, the server MUST reply with an "unknown-element" error-tag in the rpc-error.

- o If data for a node tagged with "when" is present, and the "when" condition evaluates to "false", the server MUST reply with an "unknown-element" error-tag in the rpc-error.

- o For insert handling, if the value for the attributes "before" and "after" are not valid for the type of the appropriate key leafs, the server MUST reply with a "bad-attribute" error-tag in the rpc-error.

- o If the attributes "before" and "after" appears in any element that is not a list whose "ordered-by" property is "user", the server MUST reply with an "unknown-attribute" error-tag in the rpc-error.

#### 8.3.2. NETCONF <edit-config> Processing

After the incoming data is parsed, the NETCONF server performs the <edit-config> operation by applying the data to the configuration datastore. During this processing, the following errors MUST be detected:

- o Delete requests for non-existent data.

- o Create requests for existent data.

- o Insert requests with "before" or "after" parameters that do not exist.

During <edit-config> processing:

- o If the NETCONF operation creates data nodes under a "choice", any existing nodes from other "case" branches are deleted by the server.

- o If the NETCONF operation modifies a data node such that any node’s "when" expression becomes false, then the node with the "when" expression is deleted by the server.

### 8.3.3. Validation

When datastore processing is complete, the final contents MUST obey all validation constraints. This validation processing is performed at differing times according to the datastore. If the datastore is "running" or "startup", these constraints MUST be enforced at the end of the <edit-config> or <copy-config> operation. If the datastore is "candidate", the constraint enforcement is delayed until a <commit> or <validate> operation.

- o Any "must" constraints MUST evaluate to "true".
- o Any referential integrity constraints defined via the "path" statement MUST be satisfied.
- o Any "unique" constraints on lists MUST be satisfied.
- o The "min-elements" and "max-elements" constraints are enforced for lists and leaf-lists.

## 9. Built-In Types

YANG has a set of built-in types, similar to those of many programming languages, but with some differences due to special requirements from the management information model.

Additional types may be defined, derived from those built-in types or from other derived types. Derived types may use subtyping to formally restrict the set of possible values.

The different built-in types and their derived types allow different kinds of subtyping, namely length and regular expression restrictions of strings (Section 9.4.4, Section 9.4.5) and range restrictions of numeric types (Section 9.2.4).

The lexical representation of a value of a certain type is used in the NETCONF messages and when specifying default values and numerical ranges in YANG modules.

### 9.1. Canonical Representation

For most types, there is a single canonical representation of the type’s values. Some types allow multiple lexical representations of the same value, for example, the positive integer "17" can be represented as "+17" or "17". Implementations MUST support all lexical representations specified in this document.

When a NETCONF server sends data, it MUST be in the canonical form.

Some types have a lexical representation that depends on the XML context in which they occur. These types do not have a canonical form.

## 9.2. The Integer Built-In Types

The integer built-in types are int8, int16, int32, int64, uint8, uint16, uint32, and uint64. They represent signed and unsigned integers of different sizes:

int8 represents integer values between -128 and 127, inclusively.

int16 represents integer values between -32768 and 32767, inclusively.

int32 represents integer values between -2147483648 and 2147483647, inclusively.

int64 represents integer values between -9223372036854775808 and 9223372036854775807, inclusively.

uint8 represents integer values between 0 and 255, inclusively.

uint16 represents integer values between 0 and 65535, inclusively.

uint32 represents integer values between 0 and 4294967295, inclusively.

uint64 represents integer values between 0 and 18446744073709551615, inclusively.

### 9.2.1. Lexical Representation

An integer value is lexically represented as an optional sign ("+" or "-"), followed by a sequence of decimal digits. If no sign is specified, "+" is assumed.

For convenience, when specifying a default value for an integer in a YANG module, an alternative lexical representation can be used, which represents the value in a hexadecimal or octal notation. The hexadecimal notation consists of an optional sign ("+" or "-"), the characters "0x" followed a number of hexadecimal digits, where letters may be uppercase or lowercase. The octal notation consists of an optional sign ("+" or "-"), the character "0" followed a number of octal digits.

Note that if a default value in a YANG module has a leading zero ("0"), it is interpreted as an octal number. In the XML instance

documents, an integer is always interpreted as a decimal number, and leading zeros are allowed.

Examples:

```
// legal values
+4711                       // legal positive value
4711                        // legal positive value
-123                        // legal negative value
0xf00f                      // legal positive hexadecimal value
-0xf                        // legal negative hexadecimal value
052                         // legal positive octal value

// illegal values
- 1                         // illegal intermediate space
```

### 9.2.2. Canonical Form

The canonical form of a positive integer does not include the sign "+". Leading zeros are prohibited. The value zero is represented as "0".

### 9.2.3. Restrictions

All integer types can be restricted with the "range" statement (Section 9.2.4).

### 9.2.4. The range Statement

The "range" statement, which is an optional substatement to the "type" statement, takes as an argument a range expression string. It is used to restrict integer and decimal built-in types, or types derived from those.

A range consists of an explicit value, or a lower-inclusive bound, two consecutive dots "..", and an upper-inclusive bound. Multiple values or ranges can be given, separated by "|". If multiple values or ranges are given, they all MUST be disjoint and MUST be in ascending order. If a range restriction is applied to an already range-restricted type, the new restriction MUST be equal or more limiting, that is raising the lower bounds, reducing the upper bounds, removing explicit values or ranges, or splitting ranges into multiple ranges with intermediate gaps. Each explicit value and range boundary value given in the range expression MUST match the type being restricted, or be one of the special values "min" or "max". "min" and "max" mean the minimum and maximum value accepted for the type being restricted, respectively.

The range expression syntax is formally defined by the rule "range-arg" in Section 13.

#### 9.2.4.1. The range's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| error-app-tag | 7.5.4.2 | 0..1 |
| error-message | 7.5.4.1 | 0..1 |
| reference | 7.20.4 | 0..1 |

### 9.2.5. Usage Example

```
typedef my-base-int32-type {
    type int32 {
        range "1..4 | 10..20";
    }
}

typedef my-type1 {
    type my-base-int32-type {
        // legal range restriction
        range "11..max"; // 11..20
    }
}

typedef my-type2 {
    type my-base-int32-type {
        // illegal range restriction
        range "11..100";
    }
}
```

## 9.3. The decimal64 Built-In Type

The decimal64 type represents a subset of the real numbers, which can be represented by decimal numerals. The value space of decimal64 is the set of numbers that can be obtained by multiplying a 64-bit signed integer by a negative power of ten, i.e., expressible as "i x 10^-n" where i is an integer64 and n is an integer between 1 and 18, inclusively.

### 9.3.1. Lexical Representation

A decimal64 value is lexically represented as an optional sign ("+" or "-"), followed by a sequence of decimal digits, optionally followed by a period ('.') as a decimal indicator and a sequence of decimal digits. If no sign is specified, "+" is assumed.

### 9.3.2. Canonical Form

The canonical form of a positive decimal64 does not include the sign "+". The decimal point is required. Leading and trailing zeros are prohibited, subject to the rule that there MUST be at least one digit before and after the decimal point. The value zero is represented as "0.0".

### 9.3.3. Restrictions

A decimal64 type can be restricted with the "range" statement (Section 9.2.4).

### 9.3.4. The fraction-digits Statement

The "fraction-digits" statement, which is a substatement to the "type" statement, MUST be present if the type is "decimal64". It takes as an argument an integer between 1 and 18, inclusively. It controls the size of the minimum difference between values of a decimal64 type, by restricting the value space to numbers that are expressible as "i x 10^-n" where n is the fraction-digits argument.

The following table lists the minimum and maximum value for each fraction-digit value:

| fraction-digit | min | max |
|---|---|---|
| 1 | -922337203685477580.8 | 922337203685477580.7 |
| 2 | -92233720368547758.08 | 92233720368547758.07 |
| 3 | -9223372036854775.808 | 9223372036854775.807 |
| 4 | -922337203685477.5808 | 922337203685477.5807 |
| 5 | -92233720368547.75808 | 92233720368547.75807 |
| 6 | -9223372036854.775808 | 9223372036854.775807 |
| 7 | -922337203685.4775808 | 922337203685.4775807 |
| 8 | -92233720368.54775808 | 92233720368.54775807 |
| 9 | -9223372036.854775808 | 9223372036.854775807 |
| 10 | -922337203.6854775808 | 922337203.6854775807 |
| 11 | -92233720.36854775808 | 92233720.36854775807 |
| 12 | -9223372.036854775808 | 9223372.036854775807 |
| 13 | -922337.2036854775808 | 922337.2036854775807 |
| 14 | -92233.72036854775808 | 92233.72036854775807 |
| 15 | -9223.372036854775808 | 9223.372036854775807 |
| 16 | -922.3372036854775808 | 922.3372036854775807 |
| 17 | -92.23372036854775808 | 92.23372036854775807 |
| 18 | -9.223372036854775808 | 9.223372036854775807 |

### 9.3.5. Usage Example

```
typedef my-decimal {
    type decimal64 {
        fraction-digits 2;
        range "1 .. 3.14 | 10 | 20..max";
    }
}
```

## 9.4. The string Built-In Type

The string built-in type represents human-readable strings in YANG. Legal characters are the Unicode and ISO/IEC 10646 [ISO.10646] characters, including tab, carriage return, and line feed but excluding the other C0 control characters, the surrogate blocks, and the noncharacters. The string syntax is formally defined by the rule "yang-string" in Section 13.

### 9.4.1. Lexical Representation

A string value is lexically represented as character data in the XML instance documents.

### 9.4.2. Canonical Form

The canonical form is the same as the lexical representation. No Unicode normalization is performed of string values.

### 9.4.3. Restrictions

A string can be restricted with the "length" (Section 9.4.4) and "pattern" (Section 9.4.5) statements.

### 9.4.4. The length Statement

The "length" statement, which is an optional substatement to the "type" statement, takes as an argument a length expression string. It is used to restrict the built-in types "string" and "binary" or types derived from them.

A "length" statement restricts the number of Unicode characters in the string.

A length range consists of an explicit value, or a lower bound, two consecutive dots "..", and an upper bound. Multiple values or ranges can be given, separated by "|". Length-restricting values MUST NOT be negative. If multiple values or ranges are given, they all MUST be disjoint and MUST be in ascending order. If a length restriction is applied to an already length-restricted type, the new restriction MUST be equal or more limiting, that is, raising the lower bounds, reducing the upper bounds, removing explicit length values or ranges, or splitting ranges into multiple ranges with intermediate gaps. A length value is a non-negative integer, or one of the special values "min" or "max". "min" and "max" mean the minimum and maximum length accepted for the type being restricted, respectively. An implementation is not required to support a length value larger than 18446744073709551615.

The length expression syntax is formally defined by the rule "length-arg" in Section 13.

#### 9.4.4.1. The length’s Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| error-app-tag | 7.5.4.2 | 0..1 |
| error-message | 7.5.4.1 | 0..1 |
| reference | 7.20.4 | 0..1 |

### 9.4.5. The pattern Statement

The "pattern" statement, which is an optional substatement to the "type" statement, takes as an argument a regular expression string, as defined in [XSD-TYPES]. It is used to restrict the built-in type "string", or types derived from "string", to values that match the pattern.

If the type has multiple "pattern" statements, the expressions are ANDed together, i.e., all such expressions have to match.

If a pattern restriction is applied to an already pattern-restricted type, values must match all patterns in the base type, in addition to the new patterns.

#### 9.4.5.1. The pattern’s Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| error-app-tag | 7.5.4.2 | 0..1 |
| error-message | 7.5.4.1 | 0..1 |
| modifier | 9.4.6 | 0..1 |
| reference | 7.20.4 | 0..1 |

### 9.4.6. The modifier Statement

### 9.4.7. Usage Example

With the following typedef:

```
typedef my-base-str-type {
    type string {
        length "1..255";
    }
}
```

the following refinement is legal:

```
type my-base-str-type {
    // legal length refinement
    length "11 | 42..max"; // 11 | 42..255
}
```

and the following refinement is illegal:

```
type my-base-str-type {
    // illegal length refinement
    length "1..999";
}
```

With the following type:

```
type string {
    length "0..4";
    pattern "[0-9a-fA-F]*";
}
```

the following strings match:

```
AB          // legal
9A00        // legal
```

and the following strings do not match:

```
00ABAB      // illegal, too long
xx00        // illegal, bad characters
```

With the following type:

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

the following string match:

```
enabled     // legal
```

and the following strings do not match:

```
10-mbit     // illegal, starts with a number
xml-element // illegal, starts with illegal sequence
```

## 9.5. The boolean Built-In Type

The boolean built-in type represents a boolean value.

#### 9.5.1. Lexical Representation

The lexical representation of a boolean value is a string with a value of "true" or "false". These values MUST be in lowercase.

#### 9.5.2. Canonical Form

The canonical form is the same as the lexical representation.

#### 9.5.3. Restrictions

A boolean cannot be restricted.

### 9.6. The enumeration Built-In Type

The enumeration built-in type represents values from a set of assigned names.

#### 9.6.1. Lexical Representation

The lexical representation of an enumeration value is the assigned name string.

#### 9.6.2. Canonical Form

The canonical form is the assigned name string.

#### 9.6.3. Restrictions

An enumeration can be restricted with the "enum" (Section 9.6.4) statement.

#### 9.6.4. The enum Statement

The "enum" statement, which is a substatement to the "type" statement, MUST be present if the type is "enumeration". It is repeatedly used to specify each assigned name of an enumeration type. It takes as an argument a string which is the assigned name. The string MUST NOT be empty and MUST NOT have any leading or trailing whitespace characters. The use of Unicode control codes SHOULD be avoided.

The statement is optionally followed by a block of substatements that holds detailed enum information.

All assigned names in an enumeration MUST be unique.

When an existing enumeration type is restricted, the set of assigned names in the new type MUST be a subset of the base type's set of assigned names. The value of such an assigned name MUST not be changed.

#### 9.6.4.1. The enum's Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| value | 9.6.4.2 | 0..1 |

#### 9.6.4.2. The value Statement

The "value" statement, which is optional, is used to associate an integer value with the assigned name for the enum. This integer value MUST be in the range -2147483648 to 2147483647, and it MUST be unique within the enumeration type.

If a value is not specified, then one will be automatically assigned. If the "enum" substatement is the first one defined, the assigned value is zero (0); otherwise, the assigned value is one greater than the current highest enum value (i.e., the highest enum value, implicit or explicit, prior to the current "enum" substatement in the parent "type" statement).

If the current highest value is equal to 2147483647, then an enum value MUST be specified for "enum" substatements following the one with the current highest value.

When an existing enumeration type is restricted, the "value" statement MUST either have the same value as the in the base type or not be present, in which case the value is the same as in the base type.

### 9.6.5. Usage Example

```
leaf myenum {
    type enumeration {
        enum zero;
        enum one;
        enum seven {
            value 7;
        }
    }
}
```

The lexical representation of the leaf "myenum" with value "seven" is:

```
<myenum>seven</myenum>
```

With the following typedef:

```
typedef my-base-enumeration-type {
    type enumeration {
        enum white {
            value 1;
        }
        enum yellow {
            value 2;
        }
        enum red {
            value 3;
        }
    }
}
```

the following refinement is legal:

```
type my-base-enumeration-type {
    // legal enum refinement
    enum yellow;
    enum red {
        value 3;
    }
}
```

and the following refinement is illegal:

```
type my-base-enumeration-type {
    // illegal enum refinement
    enum yellow {
        value 4; // illegal value change
    }
    enum black; // illegal addition of new name
}
```

## 9.7. The bits Built-In Type

The bits built-in type represents a bit set. That is, a bits value is a set of flags identified by small integer position numbers starting at 0. Each bit number has an assigned name.

### 9.7.1. Restrictions

A bits type cannot be restricted.

### 9.7.2. Lexical Representation

The lexical representation of the bits type is a space-separated list of the individual bit values that are set. An empty string thus represents a value where no bits are set.

### 9.7.3. Canonical Form

In the canonical form, the bit values are separated by a single space character and they appear ordered by their position (see Section 9.7.4.2).

### 9.7.4. The bit Statement

The "bit" statement, which is a substatement to the "type" statement, MUST be present if the type is "bits". It is repeatedly used to specify each assigned named bit of a bits type. It takes as an argument a string that is the assigned name of the bit. It is followed by a block of substatements that holds detailed bit information. The assigned name follows the same syntax rules as an identifier (see Section 6.2).

All assigned names in a bits type MUST be unique.

#### 9.7.4.1. The bit’s Substatements

| substatement | section | cardinality |
|---|---|---|
| description | 7.20.3 | 0..1 |
| if-feature | 7.19.2 | 0..n |
| reference | 7.20.4 | 0..1 |
| status | 7.20.2 | 0..1 |
| position | 9.7.4.2 | 0..1 |

#### 9.7.4.2. The position Statement

The "position" statement, which is optional, takes as an argument a non-negative integer value that specifies the bit’s position within a hypothetical bit field. The position value MUST be in the range 0 to 4294967295, and it MUST be unique within the bits type. The value is unused by YANG and the NETCONF messages, but is carried as a convenience to implementors.

If a bit position is not specified, then one will be automatically assigned. If the "bit" substatement is the first one defined, the assigned value is zero (0); otherwise, the assigned value is one greater than the current highest bit position (i.e., the highest bit position, implicit or explicit, prior to the current "bit" substatement in the parent "type" statement).

If the current highest bit position value is equal to 4294967295, then a position value MUST be specified for "bit" substatements following the one with the current highest position value.

### 9.7.5. Usage Example

Given the following leaf:

```
leaf mybits {
    type bits {
        bit disable-nagle {
            position 0;
        }
        bit auto-sense-speed {
            position 1;
        }
        bit ten-Mb-only {
            position 2;
        }
    }
    default "auto-sense-speed";
}
```

The lexical representation of this leaf with bit values disable-nagle and ten-Mb-only set would be:

```
<mybits>disable-nagle ten-Mb-only</mybits>
```

## 9.8. The binary Built-In Type

The binary built-in type represents any binary data, i.e., a sequence of octets.

### 9.8.1. Restrictions

A binary can be restricted with the "length" (Section 9.4.4) statement. The length of a binary value is the number of octets it contains.

### 9.8.2. Lexical Representation

Binary values are encoded with the base64 encoding scheme (see [RFC4648], Section 4).

### 9.8.3. Canonical Form

The canonical form of a binary value follows the rules in [RFC4648].

## 9.9. The leafref Built-In Type

The leafref type is used to declare a constraint on the value space of a leaf, based on a reference to a set of leaf instances in the data tree. The "path" substatement (Section 9.9.2) selects a set of leaf instances, and the leafref value space is the set of values of these leaf instances.

If the leaf with the leafref type represents configuration data, and the "require-instance" property (Section 9.9.3) is "true", the leaf it refers to MUST also represent configuration. Such a leaf puts a constraint on valid data. All such nodes MUST reference existing leaf instances or leafs with default values in use (see Section 7.6.1 and Section 7.7.2) for the data to be valid. This constraint is enforced according to the rules in Section 8.

There MUST NOT be any circular chains of leafrefs.

If the leaf that the leafref refers to is conditional based on one or more features (see Section 7.19.2), then the leaf with the leafref type MUST also be conditional based on at least the same set of features.

### 9.9.1. Restrictions

A leafref can be restricted with the "require-instance" statement (Section 9.9.3).

### 9.9.2. The path Statement

The "path" statement, which is a substatement to the "type" statement, MUST be present if the type is "leafref". It takes as an argument a string that MUST refer to a leaf or leaf-list node.

The syntax for a path argument is a subset of the XPath abbreviated syntax. Predicates are used only for constraining the values for the key nodes for list entries. Each predicate consists of exactly one equality test per key, and multiple adjacent predicates MAY be present if a list has multiple keys. The syntax is formally defined by the rule "path-arg" in Section 13.

The predicates are only used when more than one key reference is needed to uniquely identify a leaf instance. This occurs if a list has multiple keys, or a reference to a leaf other than the key in a list is needed. In these cases, multiple leafrefs are typically specified, and predicates are used to tie them together.

The "path" expression evaluates to a node set consisting of zero, one, or more nodes. If the leaf with the leafref type represents configuration data, this node set MUST be non-empty.

The "path" XPath expression is conceptually evaluated in the following context, in addition to the definition in Section 6.4.1:

- o If the "path" statement is defined within a typedef, the context node is the leaf or leaf-list node in the data tree that references the typedef.

- o Otherwise, the context node is the node in the data tree for which the "path" statement is defined.

### 9.9.3. The require-instance Statement

The "require-instance" statement, which is a substatement to the "type" statement, MAY be present if the type is "instance-identifier" or "leafref". It takes as an argument the string "true" or "false". If this statement is not present, it defaults to "true".

If "require-instance" is "true", it means that the instance being referred MUST exist for the data to be valid. This constraint is enforced according to the rules in Section 8.

If "require-instance" is "false", it means that the instance being referred MAY exist in valid data.

### 9.9.4. Lexical Representation

A leafref value is encoded the same way as the leaf it references.

### 9.9.5. Canonical Form

The canonical form of a leafref is the same as the canonical form of the leaf it references.

### 9.9.6. Usage Example

With the following list:

```
list interface {
    key "name";
    leaf name {
        type string;
    }
    leaf admin-status {
        type admin-status;
    }
    list address {
        key "ip";
        leaf ip {
            type yang:ip-address;
        }
    }
}
```

The following leafref refers to an existing interface:

```
leaf mgmt-interface {
    type leafref {
        path "../interface/name";
    }
}
```

An example of a corresponding XML snippet:

```
<interface>
  <name>eth0</name>
</interface>
<interface>
  <name>lo</name>
</interface>

<mgmt-interface>eth0</mgmt-interface>
```

The following leafrefs refer to an existing address of an interface:

```
container default-address {
    leaf ifname {
        type leafref {
            path "../../interface/name";
        }
    }
    leaf address {
        type leafref {
            path "../../interface[name = current()/../ifname]"
               + "/address/ip";
        }
    }
}
```

An example of a corresponding XML snippet:

```
<interface>
  <name>eth0</name>
  <admin-status>up</admin-status>
  <address>
    <ip>192.0.2.1</ip>
  </address>
  <address>
    <ip>192.0.2.2</ip>
  </address>
</interface>
<interface>
  <name>lo</name>
  <admin-status>up</admin-status>
  <address>
    <ip>127.0.0.1</ip>
  </address>
</interface>

<default-address>
  <ifname>eth0</ifname>
  <address>192.0.2.2</address>
</default-address>
```

The following list uses a leafref for one of its keys. This is similar to a foreign key in a relational database.

```
list packet-filter {
    key "if-name filter-id";
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf filter-id {
        type uint32;
    }
    ...
}
```

An example of a corresponding XML snippet:

```
<interface>
  <name>eth0</name>
  <admin-status>up</admin-status>
  <address>
    <ip>192.0.2.1</ip>
  </address>
  <address>
    <ip>192.0.2.2</ip>
  </address>
</interface>

<packet-filter>
  <if-name>eth0</if-name>
  <filter-id>1</filter-id>
  ...
</packet-filter>
<packet-filter>
  <if-name>eth0</if-name>
  <filter-id>2</filter-id>
  ...
</packet-filter>
```

The following notification defines two leafrefs to refer to an existing admin-status:

```
notification link-failure {
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf admin-status {
        type leafref {
            path
              "/interface[name = current()/../if-name]"
            + "/admin-status";
        }
    }
}
```

An example of a corresponding XML notification:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-04-01T00:01:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>eth0</if-name>
    <admin-status>up</admin-status>
  </link-failure>
</notification>
```

### 9.10. The identityref Built-In Type

The identityref type is used to reference an existing identity (see Section 7.17).

#### 9.10.1. Restrictions

An identityref cannot be restricted.

#### 9.10.2. The identityref's base Statement

The "base" statement, which is a substatement to the "type" statement, MUST be present at least once if the type is "identityref". The argument is the name of an identity, as defined by an "identity" statement. If a prefix is present on the identity name, it refers to an identity defined in the module that was imported with that prefix. Otherwise, an identity with the matching name MUST be defined in the current module or an included submodule.

Valid values for an identityref are any identities derived from all the identityref's base identities. On a particular server, the valid values are further restricted to the set of identities defined in the modules supported by the server.

#### 9.10.3. Lexical Representation

An identityref is encoded as the referred identity's qualified name as defined in [XML-NAMES]. If the prefix is not present, the namespace of the identityref is the default namespace in effect on the element that contains the identityref value.

When an identityref is given a default value using the "default" statement, the identity name in the default value MAY have a prefix. If a prefix is present on the identity name, it refers to an identity defined in the module that was imported with that prefix, or the prefix for the current module if the identity is defined in the current module or one of its submodules. Otherwise, an identity with the matching name MUST be defined in the current module or one of its submodules.

The string value of a node of type identityref in a "must" or "when" XPath expression is the referred identity's qualified name with the prefix present. If the referred identity is defined in an imported module, the prefix in the string value is the prefix defined in the corresponding "import" statement. Otherwise, the prefix in the string value is the prefix for the current module.

### 9.10.4. Canonical Form

Since the lexical form depends on the XML context in which the value occurs, this type does not have a canonical form.

### 9.10.5. Usage Example

With the identity definitions in Section 7.17.3 and the following module:

```
module my-crypto {
    yang-version 1.1;
    namespace "http://example.com/my-crypto";
    prefix mc;

    import "crypto-base" {
        prefix "crypto";
    }

    identity aes {
        base "crypto:crypto-alg";
    }

    leaf crypto {
        type identityref {
            base "crypto:crypto-alg";
        }
    }

    container aes-parameters {
        when "../crypto = 'mc:aes'";
        ...
    }
}
```

the following is an example how the leaf "crypto" can be encoded, if the value is the "des3" identity defined in the "des" module:

```
<crypto xmlns:des="http://example.com/des">des:des3</crypto>
```

Any prefixes used in the encoding are local to each instance encoding. This means that the same identityref may be encoded differently by different implementations. For example, the following example encodes the same leaf as above:

```
<crypto xmlns:x="http://example.com/des">x:des3</crypto>
```

If the "crypto" leaf’s value instead is "aes" defined in the "my-crypto" module, it can be encoded as:

```
<crypto xmlns:mc="http://example.com/my-crypto">mc:aes</crypto>
```

or, using the default namespace:

```
<crypto>aes</crypto>
```

### 9.11. The empty Built-In Type

The empty built-in type represents a leaf that does not have any value, it conveys information by its presence or absence.

An empty type cannot have a default value.

#### 9.11.1. Restrictions

An empty type cannot be restricted.

#### 9.11.2. Lexical Representation

Not applicable.

#### 9.11.3. Canonical Form

Not applicable.

#### 9.11.4. Usage Example

With the following leaf

```
leaf enable-qos {
    type empty;
}
```

the following is an example of a valid encoding

```
<enable-qos/>
```

if the leaf exists.

## 9.12. The union Built-In Type

The union built-in type represents a value that corresponds to one of its member types.

When the type is "union", the "type" statement (Section 7.4) MUST be present. It is used to repeatedly specify each member type of the union. It takes as an argument a string that is the name of a member type.

A member type can be of any built-in or derived type.

A value representing a union data type is validated consecutively against each member type, in the order they are specified in the "type" statement, until a match is found. The type that matched will be the type of the value for the node that was validated.

Any default value or "units" property defined in the member types is not inherited by the union type.

### 9.12.1. Restrictions

A union cannot be restricted. However, each member type can be restricted, based on the rules defined in Section 9.

### 9.12.2. Lexical Representation

The lexical representation of a union is a value that corresponds to the representation of any one of the member types.

### 9.12.3. Canonical Form

The canonical form of a union value is the same as the canonical form of the member type of the value.

### 9.12.4. Usage Example

The following is a union of an int32 an enumeration:

```
type union {
    type int32;
    type enumeration {
        enum "unbounded";
    }
}
```

Care must be taken when a member type is a leafref where the "require-instance" property (Section 9.9.3) is "true". If a leaf of

such a type refers to an existing instance, the leaf’s value must be re-validated if the target instance is deleted. For example, with the following definitions:

```
list filter {
    key name;
    leaf name {
        type string;
    }
    ...
}

leaf outbound-filter {
    type union {
        type leafref {
            path "/filter/name";
        }
        type enumeration {
            enum default-filter;
        }
    }
}
```

assume that there exists an entry in the filter list with the name "http", and that the outbound-filter leaf has this value:

```
<filter>
  <name>http</name>
</filter>
<outbound-filter>http</outbound-filter>
```

If the filter entry "http" is removed, the outbound-filter leaf’s value doesn’t match the leafref, and the next member type is checked. The current value ("http") doesn’t match the enumeration, so the resulting configuration is invalid.

If the second member type in the union had been of type "string" instead of an enumeration, the current value would have matched, and the resulting configuration would have been valid.

## 9.13. The instance-identifier Built-In Type

The instance-identifier built-in type is used to uniquely identify a particular instance node in the data tree.

The syntax for an instance-identifier is a subset of the XPath abbreviated syntax, formally defined by the rule "instance-identifier" in Section 13. It is used to uniquely identify

a node in the data tree. Predicates are used only for specifying the values for the key nodes for list entries, a value of a leaf-list entry, or a positional index for a list without keys. For identifying list entries with keys, each predicate consists of one equality test per key, and each key MUST have a corresponding predicate.

If the leaf with the instance-identifier type represents configuration data, and the "require-instance" property (Section 9.9.3) is "true", the node it refers to MUST also represent configuration. Such a leaf puts a constraint on valid data. All such leaf nodes MUST reference existing nodes or leaf or leaf-list nodes with their default value in use (see Section 7.6.1 and Section 7.7.2) for the data to be valid. This constraint is enforced according to the rules in Section 8.

The "instance-identifier" XPath expression is conceptually evaluated in the following context, in addition to the definition in Section 6.4.1:

o The context node is the root node in the accessible tree.

### 9.13.1. Restrictions

An instance-identifier can be restricted with the "require-instance" statement (Section 9.9.3).

### 9.13.2. Lexical Representation

An instance-identifier value is lexically represented as a string. All node names in an instance-identifier value MUST be qualified with explicit namespace prefixes, and these prefixes MUST be declared in the XML namespace scope in the instance-identifier’s XML element.

Any prefixes used in the encoding are local to each instance encoding. This means that the same instance-identifier may be encoded differently by different implementations.

### 9.13.3. Canonical Form

Since the lexical form depends on the XML context in which the value occurs, this type does not have a canonical form.

### 9.13.4. Usage Example

The following are examples of instance identifiers:

```
/* instance-identifier for a container */
/ex:system/ex:services/ex:ssh

/* instance-identifier for a leaf */
/ex:system/ex:services/ex:ssh/ex:port

/* instance-identifier for a list entry */
/ex:system/ex:user[ex:name='fred']

/* instance-identifier for a leaf in a list entry */
/ex:system/ex:user[ex:name='fred']/ex:type

/* instance-identifier for a list entry with two keys */
/ex:system/ex:server[ex:ip='192.0.2.1'][ex:port='80']

/* instance-identifier for a leaf-list entry */
/ex:system/ex:services/ex:ssh/ex:cipher[.='blowfish-cbc']

/* instance-identifier for a list entry without keys */
/ex:stats/ex:port[3]
```

# 10. XPath Functions

This document defines two generic XPath functions and four YANG type-specific XPath functions. The function signatures are specified with the syntax used in [XPATH].

## 10.1. Functions for Node Sets

### 10.1.1. current()

```
node-set current()
```

The function current() takes no input parameters, and returns a node set with the initial context node as its only member.

## 10.2. Functions for Strings

### 10.2.1. re-match()

```
boolean re-match(string subject, string pattern)
```

The re-match() function returns true if the "subject" string matches the regular expression "pattern"; otherwise it returns false.

The function "re-match" checks if a string matches a given regular expression. The regular expressions used are the XML Schema regular

expressions [XSD-TYPES]. Note that this includes implicit anchoring of the regular expression at the head and tail.

#### 10.2.1.1. Usage Example

The expression:

```
re-match('1.22.333', '\d{1,3}\.\d{1,3}\.\d{1,3}')
```

returns true.

To count all logical interfaces called eth0.<number>, do:

```
count(/interface[re-match(name, 'eth0\.\d+')])
```

## 10.3. Functions for the YANG Types "leafref" and "instance-identifier"

### 10.3.1. deref()

```
node-set deref(node-set nodes)
```

The deref() function follows the reference defined by the first node in document order in the argument "nodes", and returns the nodes it refers to.

If the first argument node is of type instance-identifier, the function returns a node set that contains the single node that the instance identifier refers to, if it exists. If no such node exists, an empty node-set is returned.

If the first argument node is of type leafref, the function returns a node set that contains the nodes that the leafref refers to.

If the first argument node is of any other type, an empty node set is returned.

#### 10.3.1.1. Usage Example

```
list interface {
    key name;
    leaf name { ... }
    leaf enabled {
        type boolean;
    }
    ...
}

leaf mgmt-interface {
    type leafref {
        path "/interface/name";
    }
    must 'deref(.)/../enabled = "true"' {
        error-message
            "The management interface cannot be disabled.";
    }
}
```

### 10.4. Functions for the YANG Type "identityref"

#### 10.4.1. derived-from()

```
boolean derived-from(node-set nodes,
                     string module-name,
                     string identity-name)
```

The derived-from() function returns true if the first node in document order in the argument "nodes" is a node of type identityref, and its value is an identity that is derived from the identity "identity-name" defined in the YANG module "module-name"; otherwise it returns false.

#### 10.4.2. derived-from-or-self()

```
boolean derived-from-or-self(node-set nodes,
                             string module-name,
                             string identity-name)
```

The derived-from-or-self() function returns true if the first node in document order in the argument "nodes" is a node of type identityref, and its value is an identity that is equal to or derived from the identity "identity-name" defined in the YANG module "module-name"; otherwise it returns false.

#### 10.4.2.1. Usage Example

```
module example-interface {
    ...

    identity interface-type;

    identity ethernet {
        base interface-type;
    }

    identity fast-ethernet {
        base ethernet;
    }

    identity gigabit-ethernet {
        base ethernet;
    }

    list interface {
        key name;
        ...
        leaf type {
            type identityref {
                base interface-type;
            }
        }
        ...
    }

    augment "/interface" {
        when 'derived-from(type,
                           "example-interface",
                           "ethernet")';
        // ethernet-specific definitions here
    }
}
```

## 10.5. Functions for the YANG Type "enumeration"

### 10.5.1. enum-value()

```
number enum-value(node-set nodes)
```

The enum-value() function checks if the first node in document order in the argument "nodes" is a node of type enumeration, and returns the enum's integer value. If the "nodes" node set is empty, or if the first node in "nodes" is not of type enumeration, it returns NaN.

#### 10.5.1.1. Usage Example

With this data model:

```
list alarm {
    ...
    leaf severity {
        type enumeration {
            enum cleared {
                value 1;
            }
            enum indeterminate {
                value 2;
            }
            enum minor {
                value 3;
            }
            enum warning {
                value 4;
            }
            enum major {
                value 5;
            }
            enum critical {
                value 6;
            }
        }
    }
}
```

the following XPath expression selects only alarms that are of severity "major" or higher:

```
/alarm[enum-value(severity) >= 5]
```

## 10.6. Functions for the YANG Type "bits"

### 10.6.1. bit-is-set()

```
boolean bit-is-set(node-set nodes, string bit-name)
```

The bit-is-set() function returns true if the first node in document order in the argument "nodes" is a node of type bits, and its value has the bit "’bit-name" set; otherwise it returns false.

#### 10.6.1.1. Usage Example

If an interface has this leaf:

```
leaf flags {
    type bits {
        bit UP;
        bit PROMISCUOUS
        bit DISABLED;
    }
 }
```

the following XPath expression can be used to select all interfaces with the UP flag set:

```
/interface[bit-is-set(flags, "UP")]
```

## 11. Updating a Module

As experience is gained with a module, it may be desirable to revise that module. However, changes are not allowed if they have any potential to cause interoperability problems between a client using an original specification and a server using an updated specification.

For any published change, a new "revision" statement (Section 7.1.9) MUST be included in front of the existing "revision" statements. If there are no existing "revision" statements, then one MUST be added to identify the new revision. Furthermore, any necessary changes MUST be applied to any meta-data statements, including the "organization" and "contact" statements (Section 7.1.7, Section 7.1.8).

Note that definitions contained in a module are available to be imported by any other module, and are referenced in "import" statements via the module name. Thus, a module name MUST NOT be changed. Furthermore, the "namespace" statement MUST NOT be changed, since all XML elements are qualified by the namespace.

Obsolete definitions MUST NOT be removed from modules since their identifiers may still be referenced by other modules.

A definition may be revised in any of the following ways:

- An "enumeration" type may have new enums added, provided the old enums’s values do not change.

- o A "bits" type may have new bits added, provided the old bit positions do not change.
- o A "range", "length", or "pattern" statement may expand the allowed value space.
- o A "default" statement may be added to a leaf that does not have a default value (either directly or indirectly through its type).
- o A "units" statement may be added.
- o A "reference" statement may be added or updated.
- o A "must" statement may be removed or its constraint relaxed.
- o A "mandatory" statement may be removed or changed from "true" to "false".
- o A "min-elements" statement may be removed, or changed to require fewer elements.
- o A "max-elements" statement may be removed, or changed to allow more elements.
- o A "description" statement may be added or clarified without changing the semantics of the definition.
- o A "base" statement may be added to an "identity" statement.
- o A "base" statement may be removed from an "identityref" type, provided there is at least one "base" statement left.
- o New typedefs, groupings, rpcs, notifications, extensions, features, and identities may be added.
- o New data definition statements may be added if they do not add mandatory nodes (Section 3.1) to existing nodes or at the top level in a module or submodule, or if they are conditionally dependent on a new feature (i.e., have an "if-feature" statement that refers to a new feature).
- o A new "case" statement may be added.
- o A node that represented state data may be changed to represent configuration, provided it is not mandatory (Section 3.1).
- o An "if-feature" statement may be removed, provided its node is not mandatory (Section 3.1).

- o A "status" statement may be added, or changed from "current" to "deprecated" or "obsolete", or from "deprecated" to "obsolete".

- o A "type" statement may be replaced with another "type" statement that does not change the syntax or semantics of the type. For example, an inline type definition may be replaced with a typedef, but an int8 type cannot be replaced by an int16, since the syntax would change.

- o Any set of data definition nodes may be replaced with another set of syntactically and semantically equivalent nodes. For example, a set of leafs may be replaced by a uses of a grouping with the same leafs.

- o A module may be split into a set of submodules, or a submodule may be removed, provided the definitions in the module do not change in any other way than allowed here.

- o The "prefix" statement may be changed, provided all local uses of the prefix also are changed.

Otherwise, if the semantics of any previous definition are changed (i.e., if a non-editorial change is made to any definition other than those specifically allowed above), then this MUST be achieved by a new definition with a new identifier.

In statements that have any data definition statements as substatements, those data definition substatements MUST NOT be reordered.

# 12. YIN

A YANG module can be translated into an alternative XML-based syntax called YIN. The translated module is called a YIN module. This section describes symmetric mapping rules between the two formats.

The YANG and YIN formats contain equivalent information using different notations. The YIN notation enables developers to represent YANG data models in XML and therefore use the rich set of XML-based tools for data filtering and validation, automated generation of code and documentation, and other tasks. Tools like XSLT or XML validators can be utilized.

The mapping between YANG and YIN does not modify the information content of the model. Comments and whitespace are not preserved.

## 12.1. Formal YIN Definition

There is a one-to-one correspondence between YANG keywords and YIN elements. The local name of a YIN element is identical to the corresponding YANG keyword. This means, in particular, that the document element (root) of a YIN document is always <module> or <submodule>.

YIN elements corresponding to the YANG keywords belong to the namespace whose associated URI is "urn:ietf:params:xml:ns:yang:yin:1".

YIN elements corresponding to extension keywords belong to the namespace of the YANG module where the extension keyword is declared via the "extension" statement.

The names of all YIN elements MUST be properly qualified with their namespaces specified above using the standard mechanisms of [XML-NAMES], i.e., "xmlns" and "xmlns:xxx" attributes.

The argument of a YANG statement is represented in YIN either as an XML attribute or a subelement of the keyword element. Table 1 defines the mapping for the set of YANG keywords. For extensions, the argument mapping is specified within the "extension" statement (see Section 7.18). The following rules hold for arguments:

- o If the argument is represented as an attribute, this attribute has no namespace.
- o If the argument is represented as an element, it is qualified by the same namespace as its parent keyword element.
- o If the argument is represented as an element, it MUST be the first child of the keyword element.

Substatements of a YANG statement are represented as (additional) children of the keyword element and their relative order MUST be the same as the order of substatements in YANG.

Comments in YANG MAY be mapped to XML comments.

| keyword | argument name | yin-element |
|---|---|---|
| anyxml | name | false |
| argument | name | false |
| augment | target-node | false |
| base | name | false |

| | | |
|---|---|---|
| belongs-to | module | false |
| bit | name | false |
| case | name | false |
| choice | name | false |
| config | value | false |
| contact | text | true |
| container | name | false |
| default | value | false |
| description | text | true |
| deviate | value | false |
| deviation | target-node | false |
| enum | name | false |
| error-app-tag | value | false |
| error-message | value | true |
| extension | name | false |
| feature | name | false |
| fraction-digits | value | false |
| grouping | name | false |
| identity | name | false |
| if-feature | name | false |
| import | module | false |
| include | module | false |
| input | <no argument> | n/a |
| key | value | false |
| leaf | name | false |
| leaf-list | name | false |
| length | value | false |
| list | name | false |
| mandatory | value | false |
| max-elements | value | false |
| min-elements | value | false |
| module | name | false |
| must | condition | false |
| namespace | uri | false |
| notification | name | false |
| ordered-by | value | false |
| organization | text | true |
| output | <no argument> | n/a |
| path | value | false |
| pattern | value | false |
| position | value | false |
| prefix | value | false |
| presence | value | false |
| range | value | false |
| reference | text | true |
| refine | target-node | false |
| require-instance | value | false |
| revision | date | false |

| | | |
|---|---|---|
| revision-date | date | false |
| rpc | name | false |
| status | value | false |
| submodule | name | false |
| type | name | false |
| typedef | name | false |
| unique | tag | false |
| units | name | false |
| uses | name | false |
| value | value | false |
| when | condition | false |
| yang-version | value | false |
| yin-element | value | false |

Table 1: Mapping of arguments of the YANG statements.

### 12.1.1. Usage Example

The following YANG module:

```
module acme-foo {
    yang-version 1.1;
    namespace "http://acme.example.com/foo";
    prefix "acfoo";

    import my-extensions {
        prefix "myext";
    }

    list interface {
        key "name";
        leaf name {
            type string;
        }

        leaf mtu {
            type uint32;
            description "The MTU of the interface.";
            myext:c-define "MY_MTU";
        }
    }
}
```

where the extension "c-define" is defined in Section 7.18.3, is translated into the following YIN:

```
<module name="acme-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:acfoo="http://acme.example.com/foo"
        xmlns:myext="http://example.com/my-extensions">

  <namespace uri="http://acme.example.com/foo"/>
  <prefix value="acfoo"/>

  <import module="my-extensions">
    <prefix value="myext"/>
  </import>

  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

## 13. YANG ABNF Grammar

In YANG, almost all statements are unordered. The ABNF grammar [RFC5234] [RFC7405] defines the canonical order. To improve module readability, it is RECOMMENDED that clauses be entered in this order.

Within the ABNF grammar, unordered statements are marked with comments.

This grammar assumes that the scanner replaces YANG comments with a single space character.

<CODE BEGINS> file "yang.abnf"

```
module-stmt         = optsep module-keyword sep identifier-arg-str
                      optsep
                      "{" stmtsep
                          module-header-stmts
                          linkage-stmts
                          meta-stmts
                          revision-stmts
```

```
                            body-stmts
                          "}" optsep

   submodule-stmt      = optsep submodule-keyword sep identifier-arg-str
                         optsep
                         "{" stmtsep
                             submodule-header-stmts
                             linkage-stmts
                             meta-stmts
                             revision-stmts
                             body-stmts
                         "}" optsep

   module-header-stmts = ;; these stmts can appear in any order
                         yang-version-stmt
                         namespace-stmt
                         prefix-stmt

   submodule-header-stmts =
                         ;; these stmts can appear in any order
                         yang-version-stmt
                         belongs-to-stmt

   meta-stmts          = ;; these stmts can appear in any order
                         [organization-stmt]
                         [contact-stmt]
                         [description-stmt]
                         [reference-stmt]

   linkage-stmts       = ;; these stmts can appear in any order
                         *import-stmt
                         *include-stmt

   revision-stmts      = *revision-stmt

   body-stmts          = *(extension-stmt /
                           feature-stmt /
                           identity-stmt /
                           typedef-stmt /
                           grouping-stmt /
                           data-def-stmt /
                           augment-stmt /
                           rpc-stmt /
                           notification-stmt /
                           deviation-stmt)

   data-def-stmt       = container-stmt /
                         leaf-stmt /
```

```
                          leaf-list-stmt /
                          list-stmt /
                          choice-stmt /
                          anyxml-stmt /
                          uses-stmt

   yang-version-stmt   = yang-version-keyword sep yang-version-arg-str
                         stmtend

   yang-version-arg-str = < a string that matches the rule >
                          < yang-version-arg >

   yang-version-arg    = "1.1"

   import-stmt         = import-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             prefix-stmt
                             [revision-date-stmt]
                         "}" stmtsep

   include-stmt        = include-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              [revision-date-stmt]
                          "}") stmtsep

   namespace-stmt      = namespace-keyword sep uri-str stmtend

   uri-str             = < a string that matches the rule >
                         < URI in RFC 3986 >

   prefix-stmt         = prefix-keyword sep prefix-arg-str stmtend

   belongs-to-stmt     = belongs-to-keyword sep identifier-arg-str
                         optsep
                         "{" stmtsep
                             prefix-stmt
                         "}" stmtsep

   organization-stmt   = organization-keyword sep string stmtend

   contact-stmt        = contact-keyword sep string stmtend

   description-stmt    = description-keyword sep string stmtend

   reference-stmt      = reference-keyword sep string stmtend

   units-stmt          = units-keyword sep string stmtend
```

```
revision-stmt       = revision-keyword sep revision-date optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

revision-date       = date-arg-str

revision-date-stmt  = revision-date-keyword sep revision-date stmtend

extension-stmt      = extension-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [argument-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

argument-stmt       = argument-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           [yin-element-stmt]
                       "}") stmtsep

yin-element-stmt    = yin-element-keyword sep yin-element-arg-str
                      stmtend

yin-element-arg-str = < a string that matches the rule >
                      < yin-element-arg >

yin-element-arg     = true-keyword / false-keyword

identity-stmt       = identity-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *base-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

base-stmt           = base-keyword sep identifier-ref-arg-str
```

```
                          stmtend

   feature-stmt        = feature-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *if-feature-stmt
                              [status-stmt]
                              [description-stmt]
                              [reference-stmt]
                          "}") stmtsep

   if-feature-stmt     = if-feature-keyword sep if-feature-expr-str
                         stmtend

   if-feature-expr-str = < a string that matches the rule >
                         < if-feature-expr >

   if-feature-expr     = "(" if-feature-expr ")" /
                         if-feature-expr sep boolean-operator sep
                           if-feature-expr /
                         not-keyword sep if-feature-expr /
                         identifier-ref-arg

   boolean-operator = and-keyword / or-keyword

   typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             type-stmt
                             [units-stmt]
                             [default-stmt]
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                          "}" stmtsep

   type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                         (";" /
                          "{" stmtsep
                              [type-body-stmts]
                          "}") stmtsep

   type-body-stmts     = numerical-restrictions /
                         decimal64-specification /
                         string-restrictions /
                         enum-specification /
                         leafref-specification /
```

```
                          identityref-specification /
                          instance-identifier-specification /
                          bits-specification /
                          union-specification /
                          binary-specification

   numerical-restrictions = range-stmt

   range-stmt           = range-keyword sep range-arg-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               [error-message-stmt]
                               [error-app-tag-stmt]
                               [description-stmt]
                               [reference-stmt]
                             "}") stmtsep

   decimal64-specification = ;; these stmts can appear in any order
                             fraction-digits-stmt
                             [range-stmt]

   fraction-digits-stmt = fraction-digits-keyword sep
                          fraction-digits-arg-str stmtend

   fraction-digits-arg-str = < a string that matches the rule >
                             < fraction-digits-arg >

   fraction-digits-arg = ("1" ["0" / "1" / "2" / "3" / "4" /
                               "5" / "6" / "7" / "8"])
                         / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

   string-restrictions = ;; these stmts can appear in any order
                         [length-stmt]
                         *pattern-stmt

   length-stmt          = length-keyword sep length-arg-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               [error-message-stmt]
                               [error-app-tag-stmt]
                               [description-stmt]
                               [reference-stmt]
                             "}") stmtsep

   pattern-stmt         = pattern-keyword sep string optsep
                          (";" /
```

```
                            "{" stmtsep
                                ;; these stmts can appear in any order
                                [modifier-stmt]
                                [error-message-stmt]
                                [error-app-tag-stmt]
                                [description-stmt]
                                [reference-stmt]
                             "}") stmtsep

   modifier-stmt       = modifier-keyword sep modifier-arg-str stmtend

   modifier-arg-str    = < a string that matches the rule >
                         < modifier-arg >

   modifier-arg        = invert-match-keyword

   default-stmt        = default-keyword sep string stmtend

   enum-specification  = 1*enum-stmt

   enum-stmt           = enum-keyword sep string optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *if-feature-stmt
                              [value-stmt]
                              [status-stmt]
                              [description-stmt]
                              [reference-stmt]
                            "}") stmtsep

   leafref-specification =
                         ;; these stmts can appear in any order
                         path-stmt
                         [require-instance-stmt]

   path-stmt           = path-keyword sep path-arg-str stmtend

   require-instance-stmt = require-instance-keyword sep
                           require-instance-arg-str stmtend

   require-instance-arg-str = < a string that matches the rule >
                              < require-instance-arg >

   require-instance-arg = true-keyword / false-keyword


   instance-identifier-specification =
```

```
                        [require-instance-stmt]

   identityref-specification =
                        1*base-stmt

   union-specification = 1*type-stmt

   binary-specification = [length-stmt]

   bits-specification  = 1*bit-stmt

   bit-stmt            = bit-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *if-feature-stmt
                              [position-stmt]
                              [status-stmt]
                              [description-stmt]
                              [reference-stmt]
                          "}") stmtsep

   position-stmt       = position-keyword sep
                         position-value-arg-str stmtend

   position-value-arg-str = < a string that matches the rule >
                            < position-value-arg >

   position-value-arg  = non-negative-integer-value

   status-stmt         = status-keyword sep status-arg-str stmtend

   status-arg-str      = < a string that matches the rule >
                         < status-arg >

   status-arg          = current-keyword /
                         obsolete-keyword /
                         deprecated-keyword

   config-stmt         = config-keyword sep
                         config-arg-str stmtend

   config-arg-str      = < a string that matches the rule >
                         < config-arg >

   config-arg          = true-keyword / false-keyword

   mandatory-stmt      = mandatory-keyword sep
```

```
                          mandatory-arg-str stmtend

   mandatory-arg-str   = < a string that matches the rule >
                         < mandatory-arg >

   mandatory-arg       = true-keyword / false-keyword

   presence-stmt       = presence-keyword sep string stmtend

   ordered-by-stmt     = ordered-by-keyword sep
                         ordered-by-arg-str stmtend

   ordered-by-arg-str  = < a string that matches the rule >
                         < ordered-by-arg >

   ordered-by-arg      = user-keyword / system-keyword

   must-stmt           = must-keyword sep string optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [error-message-stmt]
                              [error-app-tag-stmt]
                              [description-stmt]
                              [reference-stmt]
                            "}") stmtsep

   error-message-stmt  = error-message-keyword sep string stmtend

   error-app-tag-stmt  = error-app-tag-keyword sep string stmtend

   min-elements-stmt   = min-elements-keyword sep
                         min-value-arg-str stmtend

   min-value-arg-str   = < a string that matches the rule >
                         < min-value-arg >

   min-value-arg       = non-negative-integer-value

   max-elements-stmt   = max-elements-keyword sep
                         max-value-arg-str stmtend

   max-value-arg-str   = < a string that matches the rule >
                         < max-value-arg >

   max-value-arg       = unbounded-keyword /
                         positive-integer-value
```

```
value-stmt          = value-keyword sep integer-value-str stmtend

integer-value-str   = < a string that matches the rule >
                      < integer-value >

grouping-stmt       = grouping-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                           *action-stmt
                       "}") stmtsep

container-stmt      = container-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [presence-stmt]
                           [config-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                           *action-stmt
                       "}") stmtsep

leaf-stmt           = leaf-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          [when-stmt]
                          *if-feature-stmt
                          type-stmt
                          [units-stmt]
                          *must-stmt
                          [default-stmt]
                          [config-stmt]
                          [mandatory-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
```

```
                           "}" stmtsep

   leaf-list-stmt      = leaf-list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt]
                             *if-feature-stmt
                             type-stmt stmtsep
                             [units-stmt]
                             *must-stmt
                             *default-stmt
                             [config-stmt]
                             [min-elements-stmt]
                             [max-elements-stmt]
                             [ordered-by-stmt]
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                           "}" stmtsep

   list-stmt           = list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt]
                             *if-feature-stmt
                             *must-stmt
                             [key-stmt]
                             *unique-stmt
                             [config-stmt]
                             [min-elements-stmt]
                             [max-elements-stmt]
                             [ordered-by-stmt]
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                             *(typedef-stmt / grouping-stmt)
                             1*data-def-stmt
                             *action-stmt
                           "}" stmtsep

   key-stmt            = key-keyword sep key-arg-str stmtend

   key-arg-str         = < a string that matches the rule >
                         < key-arg >

   key-arg             = node-identifier *(sep node-identifier)

   unique-stmt         = unique-keyword sep unique-arg-str stmtend
```

```
unique-arg-str      = < a string that matches the rule >
                      < unique-arg >

unique-arg          = descendant-schema-nodeid
                      *(sep descendant-schema-nodeid)

choice-stmt         = choice-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           [default-stmt]
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(short-case-stmt / case-stmt)
                       "}") stmtsep

short-case-stmt     = choice-stmt /
                      container-stmt /
                      leaf-stmt /
                      leaf-list-stmt /
                      list-stmt /
                      anyxml-stmt

case-stmt           = case-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *data-def-stmt
                       "}") stmtsep

anyxml-stmt         = anyxml-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [config-stmt]
```

```
                                [mandatory-stmt]
                                [status-stmt]
                                [description-stmt]
                                [reference-stmt]
                             "}") stmtsep

   uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                         (";" /
                          "{" stmtsep
                                ;; these stmts can appear in any order
                                [when-stmt]
                                *if-feature-stmt
                                [status-stmt]
                                [description-stmt]
                                [reference-stmt]
                                *refine-stmt
                                *uses-augment-stmt
                          "}") stmtsep

   refine-stmt         = refine-keyword sep refine-arg-str optsep
                          "{" stmtsep
                                ;; these stmts can appear in any order
                                *if-feature-stmt
                                *must-stmt
                                [presence-stmt]
                                [default-stmt]
                                [config-stmt]
                                [mandatory-stmt]
                                [min-elements-stmt]
                                [max-elements-stmt]
                                [description-stmt]
                                [reference-stmt]
                              "}" stmtsep

   refine-arg-str      = < a string that matches the rule >
                         < refine-arg >

   refine-arg          = descendant-schema-nodeid

   uses-augment-stmt   = augment-keyword sep uses-augment-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt]
                             *if-feature-stmt
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                             1*(data-def-stmt / case-stmt / action-stmt)
```

```
                          "}" stmtsep

   uses-augment-arg-str = < a string that matches the rule >
                          < uses-augment-arg >

   uses-augment-arg    = descendant-schema-nodeid

   augment-stmt        = augment-keyword sep augment-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt]
                             *if-feature-stmt
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                             1*(data-def-stmt / case-stmt / action-stmt)
                          "}" stmtsep

   augment-arg-str     = < a string that matches the rule >
                         < augment-arg >

   augment-arg         = absolute-schema-nodeid

   when-stmt           = when-keyword sep string optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [description-stmt]
                              [reference-stmt]
                           "}") stmtsep

   rpc-stmt            = rpc-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *if-feature-stmt
                              [status-stmt]
                              [description-stmt]
                              [reference-stmt]
                              *(typedef-stmt / grouping-stmt)
                              [input-stmt]
                              [output-stmt]
                          "}") stmtsep

   action-stmt         = action-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
```

```
                                *if-feature-stmt
                                [status-stmt]
                                [description-stmt]
                                [reference-stmt]
                                *(typedef-stmt / grouping-stmt)
                                [input-stmt]
                                [output-stmt]
                             "}") stmtsep

   input-stmt           = input-keyword optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *must-stmt
                              *(typedef-stmt / grouping-stmt)
                              1*data-def-stmt
                          "}" stmtsep

   output-stmt          = output-keyword optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *must-stmt
                              *(typedef-stmt / grouping-stmt)
                              1*data-def-stmt
                          "}" stmtsep

   notification-stmt    = notification-keyword sep
                          identifier-arg-str optsep
                          (";" /
                           "{" stmtsep
                                ;; these stmts can appear in any order
                                *if-feature-stmt
                                *must-stmt
                                [status-stmt]
                                [description-stmt]
                                [reference-stmt]
                                *(typedef-stmt / grouping-stmt)
                                *data-def-stmt
                             "}") stmtsep

   deviation-stmt       = deviation-keyword sep
                          deviation-arg-str optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [description-stmt]
                              [reference-stmt]
                              (deviate-not-supported-stmt /
                                1*(deviate-add-stmt /
                                   deviate-replace-stmt /
```

```
                                 deviate-delete-stmt))
                         "}" stmtsep

   deviation-arg-str   = < a string that matches the rule >
                         < deviation-arg >

   deviation-arg       = absolute-schema-nodeid

   deviate-not-supported-stmt =
                         deviate-keyword sep
                         not-supported-keyword-str stmtend

   deviate-add-stmt    = deviate-keyword sep add-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [units-stmt]
                              *must-stmt
                              *unique-stmt
                              [default-stmt]
                              [config-stmt]
                              [mandatory-stmt]
                              [min-elements-stmt]
                              [max-elements-stmt]
                          "}") stmtsep

   deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [units-stmt]
                              *must-stmt
                              *unique-stmt
                              [default-stmt]
                          "}") stmtsep

   deviate-replace-stmt = deviate-keyword sep replace-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [type-stmt]
                              [units-stmt]
                              [default-stmt]
                              [config-stmt]
                              [mandatory-stmt]
                              [min-elements-stmt]
                              [max-elements-stmt]
                          "}") stmtsep
```

```
not-supported-keyword-str = < a string that matches the rule >
                            < not-supported-keyword >

add-keyword-str     = < a string that matches the rule >
                      < add-keyword >

delete-keyword-str  = < a string that matches the rule >
                      < delete-keyword >

replace-keyword-str = < a string that matches the rule >
                      < replace-keyword >

;; represents the usage of an extension statement
unknown-statement   = prefix ":" identifier [sep string] optsep
                      (";" /
                       "{" optsep
                           *((yang-stmt / unknown-statement) optsep)
                        "}") stmtsep

yang-stmt           = action-stmt /
                      anyxml-stmt /
                      argument-stmt /
                      augment-stmt /
                      base-stmt /
                      belongs-to-stmt /
                      bit-stmt /
                      case-stmt /
                      choice-stmt /
                      config-stmt /
                      contact-stmt /
                      container-stmt /
                      default-stmt /
                      description-stmt /
                      deviate-add-stmt /
                      deviate-delete-stmt /
                      deviate-not-supported-stmt /
                      deviate-replace-stmt /
                      deviation-stmt /
                      enum-stmt /
                      error-app-tag-stmt /
                      error-message-stmt /
                      extension-stmt /
                      feature-stmt /
                      fraction-digits-stmt /
                      grouping-stmt /
                      identity-stmt /
                      if-feature-stmt /
                      import-stmt /
```

```
                            include-stmt /
                            input-stmt /
                            key-stmt /
                            leaf-list-stmt /
                            leaf-stmt /
                            length-stmt /
                            list-stmt /
                            mandatory-stmt /
                            max-elements-stmt /
                            min-elements-stmt /
                            modifier-stmt /
                            module-stmt /
                            must-stmt /
                            namespace-stmt /
                            notification-stmt /
                            ordered-by-stmt /
                            organization-stmt /
                            output-stmt /
                            path-stmt /
                            pattern-stmt /
                            position-stmt /
                            prefix-stmt /
                            presence-stmt /
                            range-stmt /
                            reference-stmt /
                            refine-stmt /
                            require-instance-stmt /
                            revision-date-stmt /
                            revision-stmt /
                            rpc-stmt /
                            status-stmt /
                            submodule-stmt /
                            typedef-stmt /
                            type-stmt /
                            unique-stmt /
                            units-stmt /
                            uses-augment-stmt /
                            uses-stmt /
                            value-stmt /
                            when-stmt /
                            yang-version-stmt /
                            yin-element-stmt

   ;; Ranges

   range-arg-str        = < a string that matches the rule >
                          < range-arg >
```

```
range-arg           = range-part *(optsep "|" optsep range-part)

range-part          = range-boundary
                      [optsep ".." optsep range-boundary]

range-boundary      = min-keyword / max-keyword /
                      integer-value / decimal-value

;; Lengths

length-arg-str      = < a string that matches the rule >
                      < length-arg >

length-arg          = length-part *(optsep "|" optsep length-part)

length-part         = length-boundary
                      [optsep ".." optsep length-boundary]

length-boundary     = min-keyword / max-keyword /
                      non-negative-integer-value

;; Date

date-arg-str        = < a string that matches the rule >
                      < date-arg >

date-arg            = 4DIGIT "-" 2DIGIT "-" 2DIGIT

;; Schema Node Identifiers

schema-nodeid       = absolute-schema-nodeid /
                      descendant-schema-nodeid

absolute-schema-nodeid = 1*("/" node-identifier)

descendant-schema-nodeid =
                      node-identifier
                      [absolute-schema-nodeid]

node-identifier     = [prefix ":"] identifier


;; Instance Identifiers

instance-identifier = 1*("/" (node-identifier *predicate))

predicate           = "[" *WSP (predicate-expr / pos) *WSP "]"
```

```
   predicate-expr      = (node-identifier / ".") *WSP "=" *WSP
                         ((DQUOTE string DQUOTE) /
                          (SQUOTE string SQUOTE))

   pos                 = non-negative-integer-value


   ;; leafref path

   path-arg-str        = < a string that matches the rule >
                         < path-arg >

   path-arg            = absolute-path / relative-path

   absolute-path       = 1*("/" (node-identifier *path-predicate))

   relative-path       = 1*("../") descendant-path

   descendant-path     = node-identifier
                         [*path-predicate absolute-path]

   path-predicate      = "[" *WSP path-equality-expr *WSP "]"

   path-equality-expr  = node-identifier *WSP "=" *WSP path-key-expr

   path-key-expr       = current-function-invocation *WSP "/" *WSP
                         rel-path-keyexpr

   rel-path-keyexpr    = 1*("../" *WSP "/" *WSP)
                         *(node-identifier *WSP "/" *WSP)
                         node-identifier

   ;;; Keywords, using RFC 7405 syntax for case-sensitive strings

   ;; statement keywords
   action-keyword      = %s"action"
   anyxml-keyword      = %s"anyxml"
   argument-keyword    = %s"argument"
   augment-keyword     = %s"augment"
   base-keyword        = %s"base"
   belongs-to-keyword  = %s"belongs-to"
   bit-keyword         = %s"bit"
   case-keyword        = %s"case"
   choice-keyword      = %s"choice"
   config-keyword      = %s"config"
   contact-keyword     = %s"contact"
   container-keyword   = %s"container"
   default-keyword     = %s"default"
```

```
description-keyword = %s"description"
enum-keyword        = %s"enum"
error-app-tag-keyword = %s"error-app-tag"
error-message-keyword = %s"error-message"
extension-keyword   = %s"extension"
deviation-keyword   = %s"deviation"
deviate-keyword     = %s"deviate"
feature-keyword     = %s"feature"
fraction-digits-keyword = %s"fraction-digits"
grouping-keyword    = %s"grouping"
identity-keyword    = %s"identity"
if-feature-keyword  = %s"if-feature"
import-keyword      = %s"import"
include-keyword     = %s"include"
input-keyword       = %s"input"
key-keyword         = %s"key"
leaf-keyword        = %s"leaf"
leaf-list-keyword   = %s"leaf-list"
length-keyword      = %s"length"
list-keyword        = %s"list"
mandatory-keyword   = %s"mandatory"
max-elements-keyword = %s"max-elements"
min-elements-keyword = %s"min-elements"
modifier-keyword    = %s"modifier"
module-keyword      = %s"module"
must-keyword        = %s"must"
namespace-keyword   = %s"namespace"
notification-keyword= %s"notification"
ordered-by-keyword  = %s"ordered-by"
organization-keyword= %s"organization"
output-keyword      = %s"output"
path-keyword        = %s"path"
pattern-keyword     = %s"pattern"
position-keyword    = %s"position"
prefix-keyword      = %s"prefix"
presence-keyword    = %s"presence"
range-keyword       = %s"range"
reference-keyword   = %s"reference"
refine-keyword      = %s"refine"
require-instance-keyword = %s"require-instance"
revision-keyword    = %s"revision"
revision-date-keyword = %s"revision-date"
rpc-keyword         = %s"rpc"
status-keyword      = %s"status"
submodule-keyword   = %s"submodule"
type-keyword        = %s"type"
typedef-keyword     = %s"typedef"
unique-keyword      = %s"unique"
```

```
units-keyword       = %s"units"
uses-keyword        = %s"uses"
value-keyword       = %s"value"
when-keyword        = %s"when"
yang-version-keyword= %s"yang-version"
yin-element-keyword = %s"yin-element"

;; other keywords

add-keyword         = %s"add"
current-keyword     = %s"current"
delete-keyword      = %s"delete"
deprecated-keyword  = %s"deprecated"
false-keyword       = %s"false"
invert-match-keyword = %s"invert-match"
max-keyword         = %s"max"
min-keyword         = %s"min"
not-supported-keyword = %s"not-supported"
obsolete-keyword    = %s"obsolete"
replace-keyword     = %s"replace"
system-keyword      = %s"system"
true-keyword        = %s"true"
unbounded-keyword   = %s"unbounded"
user-keyword        = %s"user"

and-keyword         = %s"and"
or-keyword          = %s"or"
not-keyword         = %s"not"

current-function-invocation = current-keyword *WSP "(" *WSP ")"

;;; Basic Rules

prefix-arg-str      = < a string that matches the rule >
                      < prefix-arg >

prefix-arg          = prefix

prefix              = identifier

identifier-arg-str  = < a string that matches the rule >
                      < identifier-arg >

identifier-arg      = identifier

;; An identifier MUST NOT start with (('X'|'x') ('M'|'m') ('L'|'l'))
identifier          = (ALPHA / "_")
                      *(ALPHA / DIGIT / "_" / "-" / ".")
```

```
identifier-ref-arg-str = < a string that matches the rule >
                         < identifier-ref-arg >

identifier-ref-arg  = identifier-ref

identifier-ref      = [prefix ":"] identifier

string              = < an unquoted string as returned by >
                      < the scanner, that matches the rule >
                      < yang-string >

yang-string        = *yang-char

;; any Unicode character including tab, carriage return, and line
;; feed, but excluding the other C0 control characters, the surrogate
;; blocks, and the noncharacters.
yang-char = %x9 / %xA / %xD / %x20-D7FF /
                            ; exclude surrogate blocks %xD800-DFFF
           %xE000-FDCF /    ; exclude noncharacters %xFDD0-FDEF
           %xFDF0-FFFD /    ; exclude noncharacters %xFFFE-FFFF
           %x10000-1FFFD /  ; exclude noncharacters %x1FFFE-1FFFF
           %x20000-2FFFD /  ; exclude noncharacters %x2FFFE-2FFFF
           %x30000-3FFFD /  ; exclude noncharacters %x3FFFE-3FFFF
           %x40000-4FFFD /  ; exclude noncharacters %x4FFFE-4FFFF
           %x50000-5FFFD /  ; exclude noncharacters %x5FFFE-5FFFF
           %x60000-6FFFD /  ; exclude noncharacters %x6FFFE-6FFFF
           %x70000-7FFFD /  ; exclude noncharacters %x7FFFE-7FFFF
           %x80000-8FFFD /  ; exclude noncharacters %x8FFFE-8FFFF
           %x90000-9FFFD /  ; exclude noncharacters %x9FFFE-9FFFF
           %xA0000-AFFFD /  ; exclude noncharacters %xAFFFE-AFFFF
           %xB0000-BFFFD /  ; exclude noncharacters %xBFFFE-BFFFF
           %xC0000-CFFFD /  ; exclude noncharacters %xCFFFE-CFFFF
           %xD0000-DFFFD /  ; exclude noncharacters %xDFFFE-DFFFF
           %xE0000-EFFFD /  ; exclude noncharacters %xEFFFE-EFFFF
           %xF0000-FFFFD /  ; exclude noncharacters %xFFFFE-FFFFF
           %x100000-10FFFD  ; exclude noncharacters %x10FFFE-10FFFF

integer-value       = ("-" non-negative-integer-value)  /
                       non-negative-integer-value

non-negative-integer-value = "0" / positive-integer-value

positive-integer-value = (non-zero-digit *DIGIT)

zero-integer-value  = 1*DIGIT

stmtend             = optsep (";" / "{" stmtsep "}") stmtsep
```

```
sep                 = 1*(WSP / line-break)
                      ; unconditional separator

optsep              = *(WSP / line-break)

stmtsep             = *(WSP / line-break / unknown-statement)

line-break          = CRLF / LF

non-zero-digit      = %x31-39

decimal-value       = integer-value ("." zero-integer-value)

SQUOTE              = %x27
                      ; ' (Single Quote)

;;; RFC 5234 core rules.

ALPHA               = %x41-5A / %x61-7A
                      ; A-Z / a-z

CR                  = %x0D
                      ; carriage return

CRLF                = CR LF
                      ; Internet standard new line

DIGIT               = %x30-39
                      ; 0-9

DQUOTE              = %x22
                      ; double quote

HEXDIG              = DIGIT /
                      %x61 / %x62 / %x63 / %x64 / %x65 / %x66
                      ; only lower-case a..f

HTAB                = %x09
                      ; horizontal tab

LF                  = %x0A
                      ; linefeed

SP                  = %x20
                      ; space

VCHAR               = %x21-7E
                      ; visible (printing) characters
```

```
WSP                 = SP / HTAB
                      ; whitespace
```

<CODE ENDS>

## 14. Error Responses for YANG Related Errors

A number of NETCONF error responses are defined for error cases related to the data-model handling. If the relevant YANG statement has an "error-app-tag" substatement, that overrides the default value specified below.

### 14.1. Error Message for Data That Violates a unique Statement

If a NETCONF operation would result in configuration data where a unique constraint is invalidated, the following error is returned:

```
error-tag:      operation-failed
error-app-tag:  data-not-unique
error-info:     <non-unique>: Contains an instance identifier that
                points to a leaf that invalidates the unique
                constraint. This element is present once for each
                non-unique leaf.

                The <non-unique> element is in the YANG
                namespace ("urn:ietf:params:xml:ns:yang:1").
```

### 14.2. Error Message for Data That Violates a max-elements Statement

If a NETCONF operation would result in configuration data where a list or a leaf-list would have too many entries the following error is returned:

```
error-tag:      operation-failed
error-app-tag:  too-many-elements
```

This error is returned once, with the error-path identifying the list node, even if there are more than one extra child present.

### 14.3. Error Message for Data That Violates a min-elements Statement

If a NETCONF operation would result in configuration data where a list or a leaf-list would have too few entries the following error is returned:

```
error-tag:      operation-failed
error-app-tag:  too-few-elements
```

This error is returned once, with the error-path identifying the list node, even if there are more than one child missing.

### 14.4. Error Message for Data That Violates a must Statement

If a NETCONF operation would result in configuration data where the restrictions imposed by a "must" statement is violated the following error is returned, unless a specific "error-app-tag" substatement is present for the "must" statement.

```
error-tag:      operation-failed
error-app-tag:  must-violation
```

### 14.5. Error Message for Data That Violates a require-instance Statement

If a NETCONF operation would result in configuration data where a leaf of type "instance-identifier" marked with require-instance "true" refers to a non-existing instance, the following error is returned:

```
error-tag:      data-missing
error-app-tag:  instance-required
error-path:     Path to the instance-identifier leaf.
```

### 14.6. Error Message for Data That Does Not Match a leafref Type

If a NETCONF operation would result in configuration data where a leaf of type "leafref" refers to a non-existing instance, the following error is returned:

```
error-tag:      data-missing
error-app-tag:  instance-required
error-path:     Path to the leafref leaf.
```

### 14.7. Error Message for Data That Violates a mandatory choice Statement

If a NETCONF operation would result in configuration data where no nodes exists in a mandatory choice, the following error is returned:

```
error-tag:      data-missing
error-app-tag:  missing-choice
error-path:     Path to the element with the missing choice.
error-info:     <missing-choice>: Contains the name of the missing
                mandatory choice.

                The <missing-choice> element is in the YANG
                namespace ("urn:ietf:params:xml:ns:yang:1").
```

### 14.8. Error Message for the "insert" Operation

If the "insert" and "key" or "value" attributes are used in an <edit-config> for a list or leaf-list node, and the "key" or "value" refers to a non-existing instance, the following error is returned:

```
error-tag:      bad-attribute
error-app-tag:  missing-instance
```

## 15. IANA Considerations

This document defines a registry for YANG module and submodule names. The name of the registry is "YANG Module Names".

The registry shall record for each entry:

- o the name of the module or submodule
- o for modules, the assigned XML namespace
- o for modules, the prefix of the module
- o for submodules, the name of the module it belongs to
- o a reference to the (sub)module’s documentation (e.g., the RFC number)

There are no initial assignments.

For allocation, RFC publication is required as per RFC 5226 [RFC5226]. All registered YANG module names MUST comply with the rules for identifiers stated in Section 6.2, and MUST have a module name prefix.

The module name prefix ’ietf-’ is reserved for IETF stream documents [RFC4844], while the module name prefix ’irtf-’ is reserved for IRTF stream documents. Modules published in other RFC streams MUST have a similar suitable prefix.

All module and submodule names in the registry MUST be unique.

All XML namespaces in the registry MUST be unique.

This document registers two URIs for the YANG and YIN XML namespaces in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following have been registered.

URI: urn:ietf:params:xml:ns:yang:yin:1
URI: urn:ietf:params:xml:ns:yang:1

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces.

This document registers one capability identifier URN from the "Network Configuration Protocol (NETCONF) Capability URNs" registry:

urn:ietf:params:netconf:capability:yang-library:1.0

This document registers two new media types as defined in the following sections.

## 15.1. Media type application/yang

MIME media type name: application

MIME subtype name: yang

Mandatory parameters: none

Optional parameters: none

Encoding considerations: 8-bit

Security considerations: See Section 15 in RFC XXXX

Interoperability considerations: None

Published specification: RFC XXXX

Applications that use this media type:

YANG module validators, web servers used for downloading YANG modules, email clients, etc.

Additional information:

Magic Number: None

File Extension: .yang

Macintosh file type code: 'TEXT'

Personal and email address for further information:
Martin Bjorklund <mbj@tail-f.com>

Intended usage: COMMON

Author:
This specification is a work item of the IETF NETMOD working group, with mailing list address <netmod@ietf.org>.

Change controller:
The IESG <iesg@ietf.org>

## 15.2. Media type application/yin+xml

MIME media type name: application

MIME subtype name: yin+xml

Mandatory parameters: none

Optional parameters:

"charset": This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations:

Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: See Section 15 in RFC XXXX

Interoperability considerations: None

Published specification: RFC XXXX

Applications that use this media type:

YANG module validators, web servers used for downloading YANG modules, email clients, etc.

Additional information:

Magic Number: As specified for "application/xml" in [RFC3023], Section 3.2.

File Extension: .yin

Macintosh file type code: 'TEXT'

Personal and email address for further information:
Martin Bjorklund <mbj@tail-f.com>

Intended usage: COMMON

Author:
This specification is a work item of the IETF NETMOD working group, with mailing list address <netmod@ietf.org>.

Change controller:
The IESG <iesg@ietf.org>

## 16. Security Considerations

This document defines a language with which to write and read descriptions of management information. The language itself has no security impact on the Internet.

The same considerations are relevant as for the base NETCONF protocol (see [RFC6241], Section 9).

Data modeled in YANG might contain sensitive information. RPCs or notifications defined in YANG might transfer sensitive information.

Security issues are related to the usage of data modeled in YANG. Such issues shall be dealt with in documents describing the data models and documents about the interfaces used to manipulate the data e.g., the NETCONF documents.

Data modeled in YANG is dependent upon:

- o the security of the transmission infrastructure used to send sensitive information.
- o the security of applications that store or release such sensitive information.
- o adequate authentication and access control mechanisms to restrict the usage of sensitive data.

YANG parsers need to be robust with respect to malformed documents. Reading malformed documents from unknown or untrusted sources could result in an attacker gaining privileges of the user running the YANG parser. In an extreme situation, the entire machine could be compromised.

## 17. Contributors

The following people all contributed significantly to the initial YANG document:

- Andy Bierman (Brocade)
- Balazs Lengyel (Ericsson)
- David Partain (Ericsson)
- Juergen Schoenwaelder (Jacobs University Bremen)
- Phil Shafer (Juniper Networks)

## 18. Acknowledgements

The editor wishes to thank the following individuals, who all provided helpful comments on various versions of this document: Mehmet Ersue, Washam Fan, Joel Halpern, Leif Johansson, Ladislav Lhotka, Gerhard Muenz, Tom Petch, Randy Presuhn, David Reid, and Bert Wijnen.

## 19. ChangeLog

RFC Editor: remove this section upon publication as an RFC.

### 19.1. Version -04

- o Incorporated changes to ABNF grammar after review and errata for RFC 6020.
- o Included solution Y16-03.
- o Included solution Y49-04.
- o Included solution Y58-01.
- o Included solution Y59-01.

### 19.2. Version -03

- o Incorporated changes from WG reviews.
- o Included solution Y05-01.
- o Included solution Y10-01.
- o Included solution Y13-01.
- o Included solution Y28-02.
- o Included solution Y55-01 (parts of it was included in -01).

### 19.3. Version -02

- o Included solution Y02-01.
- o Included solution Y04-02.
- o Included solution Y11-01.
- o Included solution Y41-01.

- o Included solution Y56-01.

### 19.4. Version -01

- o Included solution Y01-01.
- o Included solution Y03-01.
- o Included solution Y06-02.
- o Included solution Y07-01.
- o Included solution Y14-01.
- o Included solution Y20-01.
- o Included solution Y23-01.
- o Included solution Y29-01.
- o Included solution Y30-01.
- o Included solution Y31-01.
- o Included solution Y35-01.

### 19.5. Version -00

- o Applied all reported errata for RFC 6020.
- o Updated YANG version to 1.1, and made the "yang-version" statement mandatory.

## 20. References

### 20.1. Normative References

[I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library (work in progress), January 2015.

[ISO.10646]
International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2003, 2003.

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

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

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

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, December 2014.

[XML-NAMES] Hollander, D., Tobin, R., Thompson, H., Bray, T., and A. Layman, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

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

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

## 20.2. Informative References

[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.

[RFC3780] Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation Structure of Management Information", RFC 3780, May 2004.

[RFC4844] Daigle, L. and Internet Architecture Board, "The RFC Series and RFC Editor", RFC 4844, July 2007.

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

[XPATH2.0]
Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0", World Wide Web Consortium Recommendation REC-xpath20-20070123, January 2007, <http://www.w3.org/TR/2007/REC-xpath20-20070123>.

[XSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xslt-19991116>.

## Author’s Address

Martin Bjorklund (editor)
Tail-f Systems

Email: mbj@tail-f.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2015

A. Bierman
YumaWorks

October 23, 2014

# Guidelines for Authors and Reviewers of YANG Data Model Documents
draft-ietf-netmod-rfc6087bis-01

## Abstract

This memo provides guidelines for authors and reviewers of Standards Track specifications containing YANG data model modules. Applicable portions may be used as a basis for reviews of other YANG data model documents. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) implementations that utilize YANG data model modules.

## Status of this Memo



## Copyright Notice

Table of Contents

## 1. Introduction

The standardization of network configuration interfaces for use with the Network Configuration Protocol [RFC6241] requires a modular set of data models, which can be reused and extended over time.

This document defines a set of usage guidelines for Standards Track documents containing [RFC6020] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF server. A server that supports a particular YANG module will support client NETCONF operation requests, as indicated by the specific content defined in the YANG module.

This document is similar to the Structure of Management Information version 2 (SMIv2) usage guidelines specification [RFC4181] in intent and structure. However, since that document was written a decade after SMIv2 modules had been in use, it was published as a 'Best Current Practice' (BCP). This document is not a BCP, but rather an informational reference, intended to promote consistency in documents containing YANG modules.

Many YANG constructs are defined as optional to use, such as the description statement. However, in order to maximize interoperability of NETCONF implementations utilizing YANG data models, it is desirable to define a set of usage guidelines that may require a higher level of compliance than the minimum level defined in the YANG specification.

In addition, YANG allows constructs such as infinite length identifiers and string values, or top-level mandatory nodes, that a compliant server is not required to support. Only constructs that all servers are required to support can be used in IETF YANG modules.

This document defines usage guidelines related to the NETCONF operations layer and NETCONF content layer, as defined in [RFC6241]. These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.

## 2. Terminology

### 2.1. Requirements Notation

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

RFC 2119 language is used here to express the views of the NETMOD working group regarding content for YANG modules. YANG modules complying with this document will treat the RFC 2119 terminology as if it were describing best current practices.

### 2.2. NETCONF Terms

The following terms are defined in [RFC6241] and are not redefined here:

- o capabilities
- o client
- o operation
- o server

### 2.3. YANG Terms

The following terms are defined in [RFC6020] and are not redefined here:

- o data node
- o module
- o namespace
- o submodule
- o version
- o YANG
- o YIN

Note that the term ’module’ may be used as a generic term for a YANG module or submodule. When describing properties that are specific to submodules, the term ’submodule’ is used instead.

## 2.4. Terms

The following terms are used throughout this document:

- published: A stable release of a module or submodule, usually contained in an RFC.

- unpublished: An unstable release of a module or submodule, usually contained in an Internet-Draft.

## 3. YANG Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module to help readers understand the module structure.

The meaning of the symbols in YANG tree diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

## 4. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors MUST be followed. The RFC Editor provides guidelines for authors of RFCs, which are first published as Internet-Drafts. These guidelines should be followed and are defined in [RFC2223] and updated in [RFC5741] and "RFC Document Style" [RFC-STYLE].

The following sections MUST be present in an Internet-Draft containing a module:

- o Narrative sections
- o Definitions section
- o Security Considerations section
- o IANA Considerations section
- o References section

### 4.1. Module Copyright

The module description statement MUST contain a reference to the latest approved IETF Trust Copyright statement, which is available online at:

http://trustee.ietf.org/license-info/

Each YANG module or submodule contained within an Internet-Draft or RFC is considered to be a code component. The strings '<CODE BEGINS>' and '<CODE ENDS>' MUST be used to identify each code component.

The '<CODE BEGINS>' tag SHOULD be followed by a string identifying the file name specified in Section 5.2 of [RFC6020]. The following example is for the '2010-01-18' revision of the 'ietf-foo' module:

```
<CODE BEGINS> file "ietf-foo@2010-01-18.yang"
  module ietf-foo {
    // ...
    revision 2010-01-18 {
      description "Latest revision";
      reference "RFC XXXX";
    }
    // ...
  }
```

<CODE ENDS>

## 4.2. Terminology Section

A terminology section MUST be present if any terms are defined in the document or if any terms are imported from other documents.

If YANG tree diagrams are used, then a sub-section explaining the YANG tree diagram syntax MUST be present, containing the following text:

> A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is defined in [RFCXXXX].
>
> -- RFC Editor: Replace XXXX with RFC number and remove note

## 4.3. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module, and SHOULD be included to help readers understand YANG module structure. Tree diagrams MAY be split into sections to correspond to document structure.

The following example shows a simple YANG tree diagram:

```
+--rw top-level-config-container
|  +--rw config-list* [key-name]
|     +--rw key-name                    string
|     +--rw optional-parm?              string
|     +--rw mandatory-parm              identityref
|     +--ro read-only-leaf              string
+--ro top-level-nonconfig-container
   +--ro nonconfig-list* [name]
      +--ro name               string
      +--ro type               string
```

## 4.4. Narrative Sections

The narrative part MUST include an overview section that describes the scope and field of application of the module(s) defined by the specification and that specifies the relationship (if any) of these modules to other standards, particularly to standards containing other YANG modules. The narrative part SHOULD include one or more sections to briefly describe the structure of the modules defined in the specification.

If the module(s) defined by the specification imports definitions

from other modules (except for those defined in the [RFC6020] or [RFC6991] documents), or are always implemented in conjunction with other modules, then those facts MUST be noted in the overview section, as MUST be noted any special interpretations of definitions in other modules.

### 4.5. Definitions Section

This section contains the module(s) defined by the specification. These modules MUST be written using the YANG syntax defined in [RFC6020]. A YIN syntax version of the module MAY also be present in the document. There MAY also be other types of modules present in the document, such as SMIv2, which are not affected by these guidelines.

See Section 5 for guidelines on YANG usage.

### 4.6. Security Considerations Section

Each specification that defines one or more modules MUST contain a section that discusses security considerations relevant to those modules.

This section MUST be patterned after the latest approved template (available at http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines). Section 7.1 contains the security considerations template dated 2010-06-16. Authors MUST check the webpage at the URL listed above in case there is a more recent version available.

In particular:

- o Writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be explained.

- o Readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

- o Operations (i.e., YANG ’rpc’ statements) that are potentially harmful to system behavior or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

### 4.7. IANA Considerations Section

In order to comply with IESG policy as set forth in http://www.ietf.org/id-info/checklist.html, every Internet-Draft that is submitted to the IESG for publication MUST contain an IANA Considerations section. The requirements for this section vary depending on what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section stating that there are no actions is removed by the RFC Editor before publication. Refer to the guidelines in [RFC5226] for more details.

#### 4.7.1. Documents that Create a New Namespace

If an Internet-Draft defines a new namespace that is to be administered by the IANA, then the document MUST include an IANA Considerations section that specifies how the namespace is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new YANG Namespace registry entry MUST be requested from the IANA. The [RFC6020] specification includes the procedure for this purpose in its IANA Considerations section.

#### 4.7.2. Documents that Extend an Existing Namespace

It is possible to extend an existing namespace using a YANG submodule that belongs to an existing module already administered by IANA. In this case, the document containing the main module MUST be updated to use the latest revision of the submodule.

### 4.8. Reference Sections

For every import or include statement that appears in a module contained in the specification, which identifies a module in a separate document, a corresponding normative reference to that document MUST appear in the Normative References section. The reference MUST correspond to the specific module version actually used within the specification.

For every normative reference statement that appears in a module contained in the specification, which identifies a separate document, a corresponding normative reference to that document SHOULD appear in the Normative References section. The reference SHOULD correspond to the specific document version actually used within the specification. If the reference statement identifies an informative reference, which identifies a separate document, a corresponding informative reference

to that document MAY appear in the Informative References section.

## 5. YANG Usage Guidelines

In general, modules in IETF Standards Track specifications MUST comply with all syntactic and semantic requirements of YANG [RFC6020]. The guidelines in this section are intended to supplement the YANG specification, which is intended to define a minimum set of conformance requirements.

In order to promote interoperability and establish a set of practices based on previous experience, the following sections establish usage guidelines for specific YANG constructs.

Only guidelines that clarify or restrict the minimum conformance requirements are included here.

### 5.1. Module Naming Conventions

Modules contained in Standards Track documents SHOULD be named according to the guidelines in the IANA Considerations section of [RFC6020].

A distinctive word or acronym (e.g., protocol name or working group acronym) SHOULD be used in the module name. If new definitions are being defined to extend one or more existing modules, then the same word or acronym should be reused, instead of creating a new one.

All published module names MUST be unique. For a YANG module published in an RFC, this uniqueness is guaranteed by IANA. For unpublished modules, the authors need to check that no other work in progress is using the same module name.

Once a module name is published, it MUST NOT be reused, even if the RFC containing the module is reclassified to 'Historic' status.

### 5.2. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an 'identifier-arg-str' token in the ABNF in Section 12 of [RFC6020].

### 5.3. Defaults

In general, it is suggested that substatements containing very common default values SHOULD NOT be present. The following substatements are commonly used with the default value, which would make the module difficult to read if used everywhere they are allowed.

| Statement | Default Value |
|---|---|
| config | true |
| mandatory | false |
| max-elements | unbounded |
| min-elements | 0 |
| ordered-by | system |
| status | current |
| yin-element | false |

Statement Defaults

## 5.4. Conditional Statements

A module may be conceptually partitioned in several ways, using the 'if-feature' and/or 'when' statements.

Data model designers need to carefully consider all modularity aspects, including the use of YANG conditional statements.

If a data definition is optional, depending on server support for a NETCONF protocol capability, then a YANG 'feature' statement SHOULD be defined to indicate that the NETCONF capability is supported within the data model.

If any notification data, or any data definition, for a non-configuration data node is not mandatory, then the server may or may not be required to return an instance of this data node. If any conditional requirements exist for returning the data node in a notification payload or retrieval request, they MUST be documented somewhere. For example, a 'when' or 'if-feature' statement could apply to the data node, or the conditional requirements could be explained in a 'description' statement within the data node or one of its ancestors (if any).

## 5.5. XPath Usage

This section describes guidelines for using the XML Path Language [W3C.REC-xpath-19991116] (XPath) within YANG modules.

### 5.5.1. Function Library

The 'position' and 'last' functions SHOULD NOT be used. This applies to implicit use of the 'position' function as well (e.g., '//chapter[42]'). A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list

or leaf-list. The 'position' and 'last' functions MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

The 'id' function SHOULD NOT be used. The 'ID' attribute is not present in YANG documents so this function has no meaning. The YANG compiler SHOULD return an empty string for this function.

The 'namespace-uri' and 'name' functions SHOULD NOT be used. Expanded names in XPath are different than YANG. A specific canonical representation of a YANG expanded name does not exist.

The 'lang' function SHOULD NOT be used. This function does not apply to YANG because there is no 'lang' attribute set with the document. The YANG compiler SHOULD return 'false' for this function.

The 'local-name', 'namespace-uri', 'name', 'string', and 'number' functions SHOULD NOT be used if the argument is a node-set. If so, the function result will be determined by the document order of the node-set. Since this order can be different on each server, the function results can also be different. Any function call that implicitly converts a node-set to a string will also have this issue.

### 5.5.2. Axes

The 'attribute' and 'namespace' axes are not supported in YANG, and MAY be empty in a NETCONF server implementation.

The 'preceding', and 'following' axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF server configuration database, which may not be supported consistently or produce reliable results across implementations. Predicate expressions based on static node properties (e.g., element name or value, 'ancestor' or 'descendant' axes) SHOULD be used instead. The 'preceding' and 'following' axes MAY be used if document order is not relevant to the outcome of the expression (e.g., check for global uniqueness of a parameter value).

The 'preceding-sibling' and 'following-sibling' axes SHOULD NOT used.

A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'preceding-sibling' and 'following-sibling' axes MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

### 5.5.3. Types

Data nodes that use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric or boolean expressions. There are boundary conditions in which the translation from the YANG 64-bit type to an XPath number can cause incorrect results. Specifically, an XPath 'double' precision floating point number cannot represent very large positive or negative 64-bit numbers because it only provides a total precision of 53 bits. The 'int64' and 'uint64' data types MAY be used in numeric expressions if the value can be represented with no more than 53 bits of precision.

Data modelers need to be careful not to confuse the YANG value space and the XPath value space. The data types are not the same in both, and conversion between YANG and XPath data types SHOULD be considered carefully.

Explicit XPath data type conversions MAY be used (e.g., 'string', 'boolean', or 'number' functions), instead of implicit XPath data type conversions.

XPath expressions that contain a literal value representing a YANG identity SHOULD always include the declared prefix of the module where the identity is defined.

XPath expressions for 'when' statements SHOULD NOT reference the context node or any descendant nodes of the context node. They MAY reference descendant nodes if the 'when' statement is contained within an 'augment' statement, and the referenced nodes are not defined within the 'augment' statement.

Example:

```
augment "/rt:active-route/rt:input/rt:destination-address" {
   when "rt:address-family='v4ur:ipv4-unicast'" {
     description
       "This augment is valid only for IPv4 unicast.";
   }
   // nodes defined here within the augment-stmt
   // cannot be referenced in the when-stmt
}
```

### 5.5.4. Wildcards

It is possible to construct XPath expressions that will evaluate differently when combined with several modules within a server implementation, then when evaluated within the single module. This is due to augmenting nodes from other modules.

Wildcard expansion is done within a server against all the nodes from all namespaces, so it is possible for a 'must' or 'when' expression that uses the '*' operator will always evaluate to false if processed within a single YANG module. In such cases, the 'description' statement SHOULD clarify that augmenting objects are expected to match the wildcard expansion.

```
when /foo/services/*/active {
  description
    "No services directly defined in this module.
     Matches objects that have augmented the services container.";
}
```

## 5.6. Lifecycle Management

The status statement MUST be present if its value is 'deprecated' or 'obsolete'. The status SHOULD NOT be changed from 'current' directly to 'obsolete'. An object SHOULD be available for at least one year with 'deprecated' status before it is changed to 'obsolete'.

The module or submodule name MUST NOT be changed, once the document containing the module or submodule is published.

The module namespace URI value MUST NOT be changed, once the document containing the module is published.

The revision-date substatement within the imports statement SHOULD be present if any groupings are used from the external module.

The revision-date substatement within the include statement SHOULD be present if any groupings are used from the external submodule.

If submodules are used, then the document containing the main module MUST be updated so that the main module revision date is equal or more recent than the revision date of any submodule that is (directly or indirectly) included by the main module.

## 5.7. Module Header, Meta, and Revision Statements

For published modules, the namespace MUST be a globally unique URI, as defined in [RFC3986]. This value is usually assigned by the IANA.

The organization statement MUST be present. If the module is contained in a document intended for Standards Track status, then the organization SHOULD be the IETF working group chartered to write the document.

The contact statement MUST be present. If the module is contained in

a document intended for Standards Track status, then the working group web and mailing information MUST be present, and the main document author or editor contact information SHOULD be present. If additional authors or editors exist, their contact information MAY be present. In addition, the Area Director and other contact information MAY be present.

The description statement MUST be present. The appropriate IETF Trust Copyright text MUST be present, as described in Section 4.1.

If the module relies on information contained in other documents, which are not the same documents implied by the import statements present in the module, then these documents MUST be identified in the reference statement.

A revision statement MUST be present for each published version of the module. The revision statement MUST have a reference substatement. It MUST identify the published document that contains the module. Modules are often extracted from their original documents, and it is useful for developers and operators to know how to find the original source document in a consistent manner. The revision statement MAY have a description substatement.

Each new revision MUST include a revision date that is higher than any other revision date in the module. The revision date does not need to be updated if the module contents do not change in the new document revision.

It is acceptable to reuse the same revision statement within unpublished versions (i.e., Internet-Drafts), but the revision date MUST be updated to a higher value each time the Internet-Draft is re-posted.

### 5.8. Namespace Assignments

It is RECOMMENDED that only valid YANG modules be included in documents, whether or not they are published yet. This allows:

- o the module to compile correctly instead of generating disruptive fatal errors.
- o early implementors to use the modules without picking a random value for the XML namespace.
- o early interoperability testing since independent implementations will use the same XML namespace value.

Until a URI is assigned by the IANA, a proposed namespace URI MUST be

provided for the namespace statement in a YANG module. A value SHOULD be selected that is not likely to collide with other YANG namespaces. Standard module names, prefixes, and URI strings already listed in the YANG Module Registry MUST NOT be used.

A standard namespace statement value SHOULD have the following form:

```
<URN prefix string>:<module-name>
```

The following URN prefix string SHOULD be used for published and unpublished YANG modules:

```
urn:ietf:params:xml:ns:yang:
```

The following example URNs would be valid temporary namespace statement values for Standards Track modules:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-partial-lock

urn:ietf:params:xml:ns:yang:ietf-netconf-state

urn:ietf:params:xml:ns:yang:ietf-netconf
```

Note that a different URN prefix string SHOULD be used for non-Standards-Track modules. The string SHOULD be selected according to the guidelines in [RFC6020].

The following examples of non-Standards-Track modules are only suggestions. There are no guidelines for this type of URN in this document:

```
http://example.com/ns/example-interfaces

http://example.com/ns/example-system
```

## 5.9. Top-Level Data Definitions

The top-level data organization SHOULD be considered carefully, in advance. Data model designers need to consider how the functionality for a given protocol or protocol family will grow over time.

The separation of configuration data and operational state SHOULD be considered carefully. It is often useful to define separate top-level containers for configuration and non-configuration data. There SHOULD only be one top-level data node defined in each YANG module for all configuration data nodes, if any configuration data nodes are defined at all. There MAY be one top-level data node defined in each YANG module for all non-configuration data nodes, if any non-

configuration data nodes are defined at all.

The names and data organization SHOULD reflect persistent information, such as the name of a protocol. The name of the working group SHOULD NOT be used because this may change over time.

A mandatory database data definition is defined as a node that a client must provide for the database to be valid. The server is not required to provide a value.

Top-level database data definitions MUST NOT be mandatory. If a mandatory node appears at the top level, it will immediately cause the database to be invalid. This can occur when the server boots or when a module is loaded dynamically at runtime.

## 5.10. Data Types

Selection of an appropriate data type (i.e., built-in type, existing derived type, or new derived type) is very subjective, and therefore few requirements can be specified on that subject.

Data model designers SHOULD use the most appropriate built-in data type for the particular application.

If extensibility of enumerated values is required, then the 'identityref' data type SHOULD be used instead of an enumeration or other built-in type.

For string data types, if a machine-readable pattern can be defined for the desired semantics, then one or more pattern statements SHOULD be present. A single quoted string SHOULD be used to specify the pattern, since a double-quoted string can modify the content.

For string data types, if the length of the string is required to be bounded in all implementations, then a length statement MUST be present.

For numeric data types, if the values allowed by the intended semantics are different than those allowed by the unbounded intrinsic data type (e.g., 'int32'), then a range statement SHOULD be present.

The signed numeric data types (i.e., 'int8', 'int16', 'int32', and 'int64') SHOULD NOT be used unless negative values are allowed for the desired semantics.

For 'enumeration' or 'bits' data types, the semantics for each 'enum' or 'bit' SHOULD be documented. A separate description statement (within each 'enum' or 'bit' statement) SHOULD be present.

## 5.11. Reusable Type Definitions

If an appropriate derived type exists in any standard module, such as [RFC6991], then it SHOULD be used instead of defining a new derived type.

If an appropriate units identifier can be associated with the desired semantics, then a units statement SHOULD be present.

If an appropriate default value can be associated with the desired semantics, then a default statement SHOULD be present.

If a significant number of derived types are defined, and it is anticipated that these data types will be reused by multiple modules, then these derived types SHOULD be contained in a separate module or submodule, to allow easier reuse without unnecessary coupling.

The description statement MUST be present.

If the type definition semantics are defined in an external document (other than another YANG module indicated by an import statement), then the reference statement MUST be present.

## 5.12. Data Definitions

The description statement MUST be present in the following YANG statements:

- o anyxml
- o augment
- o choice
- o container
- o extension
- o feature
- o grouping
- o identity
- o leaf
- o leaf-list

- o list
- o notification
- o rpc
- o typedef

If the data definition semantics are defined in an external document, (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

The 'anyxml' construct may be useful to represent an HTML banner containing markup elements, such as '<b>' and '</b>', and MAY be used in such cases. However, this construct SHOULD NOT be used if other YANG data node types can be used instead to represent the desired syntax and semantics.

If there are referential integrity constraints associated with the desired semantics that can be represented with XPath, then one or more 'must' statements SHOULD be present.

For list and leaf-list data definitions, if the number of possible instances is required to be bounded for all implementations, then the max-elements statements SHOULD be present.

If any 'must' or 'when' statements are used within the data definition, then the data definition description statement SHOULD describe the purpose of each one.

## 5.13. Operation Definitions

If the operation semantics are defined in an external document (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

If the operation impacts system behavior in some way, it SHOULD be mentioned in the description statement.

If the operation is potentially harmful to system behavior in some way, it MUST be mentioned in the Security Considerations section of the document.

## 5.14. Notification Definitions

The description statement MUST be present.

If the notification semantics are defined in an external document

(other than another YANG module indicated by an import statement), then a reference statement MUST be present.

If the notification refers to a specific resource instance, then this instance SHOULD be identified in the notification data. This is usually done by including 'leafref' leaf nodes with the key leaf values for the resource instance. For example:

```
notification interface-up {
  description "Sent when an interface is activated.";
  leaf name {
    type leafref {
      path "/if:interfaces/if:interface/if:name";
    }
  }
}
```

## 6. IANA Considerations

This document registers one URI in the IETF XML registry [RFC3688].

The following registration has been made:

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

Registrant Contact: The NETMOD WG of the IETF.

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

Per this document, the following assignment has been made in the YANG Module Names Registry for the YANG module template in Appendix C.

| Field | Value |
|---|---|
| Name | ietf-template |
| Namespace | urn:ietf:params:xml:ns:yang:ietf-template |
| Prefix | temp |
| Reference | RFC XXXX |

YANG Registry Assignment

## 7. Security Considerations

This document defines documentation guidelines for NETCONF content defined with the YANG data modeling language. The guidelines for how to write a Security Considerations section for a YANG module are defined in the online document

http://trac.tools.ietf.org/area/ops/trac/wiki/
yang-security-guidelines

This document does not introduce any new or increased security risks into the management system.

The following section contains the security considerations template dated 2010-06-16. Be sure to check the webpage at the URL listed above in case there is a more recent version available.

Each specification that defines one or more YANG modules MUST contain a section that discusses security considerations relevant to those modules. This section MUST be patterned after the latest approved template (available at

http://www.ops.ietf.org/netconf/yang-security-considerations.txt).

In particular, writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be spelled out.

Similarly, readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

Further, if new RPC operations have been defined, then the security considerations of each new RPC operation MUST be explained.

### 7.1. Security Considerations Section Template

X. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242].

```
-- if you have any writable data nodes (those are all the
-- "config true" nodes, and remember, that is the default)
-- describe their specific sensitivity or vulnerability.
```

There are a number of data nodes defined in this YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

```
 <list subtrees and data nodes and state why they are sensitive>


 -- for all YANG modules you must evaluate whether any readable data
 -- nodes (those are all the "config false" nodes, but also all other
 -- nodes, because they can also be read via operations like get or
 -- get-config) are sensitive or vulnerable (for instance, if they
 -- might reveal customer information or violate personal privacy
 -- laws such as those of the European Union if exposed to
 -- unauthorized parties)
```

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

```
 <list subtrees and data nodes and state why they are sensitive>


 -- if your YANG module has defined any rpc operations
 -- describe their specific sensitivity or vulnerability.
```

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

```
 <list RPC operations and state why they are sensitive>
```

## 8. Acknowledgments

The structure and contents of this document are adapted from [RFC4181], guidelines for MIB Documents, by C. M. Heard.

The working group thanks Martin Bjorklund, Juergen Schoenwaelder, Ladislav Lhotka, and Jernej Tuljak for their extensive reviews and contributions to this document.

## 9. Changes Since RFC 6087

The following changes have been made to the guidelines published in [RFC6087]:

- Updated NETCONF reference from RFC 4741 to RFC 6241
- Updated NETCONF over SSH citation from RFC 4742 to RFC 6242
- Updated YANG Types reference from RFC 6021 to RFC 6991
- Updated obsolete URLs for IETF resources
- Changed top-level data node guideline
- Clarified XPath usage for a literal value representing a YANG identity
- Clarified XPath usage for a when-stmt
- Added terminology guidelines
- Added YANG tree diagram definition and guideline
- Updated XPath guidelines for type conversions and function library usage.
- Updated data types section
- Updated notifications section

# 10. References

## 10.1. Normative References

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

[RFC2223] Postel, J. and J. Reynolds, "Instructions to RFC Authors", RFC 2223, October 1997.

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

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

[RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.

[RFC5741] Daigle, L., Kolkman, O., and IAB, "RFC Streams, Headers, and Boilerplates", RFC 5741, December 2009.

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

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

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[W3C.REC-xpath-19991116]
Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

## 10.2. Informative References

[RFC-STYLE]
Braden, R., Ginoza, S., and A. Hagens, "RFC Document Style", September 2009, <http://www.rfc-editor.org/rfc-style-guide/rfc-style>.

[RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB

Documents", BCP 111, RFC 4181, September 2005.

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

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

## Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

### A.1. 00 to 01

All issues from the issue tracker have been addressed.

https://github.com/netmod-wg/rfc6087bis/issues

- Issue 1: Tree Diagrams: Added Section 3 so RFCs with YANG modules can use an Informative reference to this RFC for tree diagrams. Updated guidelines to reference this RFC when tree diagrams are used

- Issue 2: XPath function restrictions: Added paragraphs in XPath usage section for 'id', 'namesapce-uri', 'name', and 'lang' functions

- Issue 3: XPath function document order issues: Added paragraph in XPath usage section about node-set ordering for 'local-name', 'namespace-uri', 'name', 'string' and 'number' functions. Also any function that implicitly converts a node-set to a string.

- Issue 4: XPath preceding-sibling and following-sibling: Checked and text in XPath usage section already has proposed text from Lada.

- Issue 5: XPath 'when-stmt' reference to descendant nodes: Added exception and example in XPath Usage section for augmented nodes.

- Issue 6: XPath numeric conversions: Changed 'numeric expressions' to 'numeric and boolean expressions'

- Issue 7: XPath module containment: Added sub-section on XPath wildcards

- Issue 8: status-stmt usage: Added text to Lifecycle Management section about transitioning from active to deprecated and then to obsolete.

- Issue 9: resource identification in notifications: Add text to Notifications section about identifying resources and using the leafref data type.

- Issue 10: single quoted strings: Added text to Data Types section about using a single-quoted string for patterns.

## Appendix B. Module Review Checklist

This section is adapted from RFC 4181.

The purpose of a YANG module review is to review the YANG module both for technical correctness and for adherence to IETF documentation requirements. The following checklist may be helpful when reviewing an Internet-Draft:

- o I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see http://www.ietf.org/id-info/guidelines.html), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.

- o Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in http://www.ietf.org/id-info/guidelines.html.

- o Copyright Notice -- verify that the draft has the appropriate text regarding the rights that document contributers provide to the IETF Trust [RFC5378]. Verify that it contains the full IETF Trust copyright notice at the beginning of the document. The IETF Trust Legal Provisions (TLP) can be found at:

  http://trustee.ietf.org/license-info/

- o Security Considerations section -- verify that the draft uses the latest approved template from the OPS area website (http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines) and that the guidelines therein have been followed.

- o IANA Considerations section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

  XML Namespace Registry: Register the YANG module namespace.

  YANG Module Registry: Register the YANG module name, prefix, namespace, and RFC number, according to the rules specified in [RFC6020].

- o References -- verify that the references are properly divided between normative and informative references, that RFC 2119 is included as a normative reference if the terminology defined therein is used in the document, that all references required by the boilerplate are present, that all YANG modules containing imported items are cited as normative references, and that all citations point to the most current RFCs unless there is a valid reason to do otherwise (for example, it is OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module).

- o License -- verify that the draft contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in Section 4.1. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

  http://trustee.ietf.org/license-info/

- o Other Issues -- check for any issues mentioned in http://www.ietf.org/id-info/checklist.html that are not covered elsewhere.

- o Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

  http://trac.tools.ietf.org/wg/netconf/trac/wiki

  Checking for correct syntax, however, is only part of the job. It is just as important to actually read the YANG module document from the point of view of a potential implementor. It is particularly important to check that description statements are sufficiently clear and unambiguous to allow interoperable implementations to be created.

## Appendix C. YANG Module Template

```
    <CODE BEGINS> file "ietf-template@2010-05-18.yang"


      module ietf-template {

        // replace this string with a unique namespace URN value
        namespace
          "urn:ietf:params:xml:ns:yang:ietf-template";

        // replace this string, and try to pick a unique prefix
        prefix "temp";


        // import statements here: e.g.,
        // import ietf-yang-types { prefix yang; }
        // import ietf-inet-types { prefix inet; }

        // identify the IETF working group if applicable
        organization
           "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

        // update this contact statement with your info
        contact
           "WG Web:   <http://tools.ietf.org/wg/your-wg-name/>
            WG List:  <mailto:your-wg-name@ietf.org>

            WG Chair: your-WG-chair
                      <mailto:your-WG-chair@example.com>

            Editor:   your-name
                      <mailto:your-email@example.com>";


        // replace the first sentence in this description statement.
        // replace the copyright notice with the most recent
        // version, if it has been updated since the publication
        // of this document
        description
         "This module defines a template for other YANG modules.

          Copyright (c) <insert year> IETF Trust and the persons
          identified as authors of the code.  All rights reserved.

          Redistribution and use in source and binary forms, with or
          without modification, is permitted pursuant to, and subject
          to the license terms contained in, the Simplified BSD License
```

```
          set forth in Section 4.c of the IETF Trust's Legal Provisions
          Relating to IETF Documents
          (http://trustee.ietf.org/license-info).

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

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

        reference "RFC XXXX";

        // RFC Ed.: remove this note
        // Note: extracted from RFC XXXX


        // replace '2010-05-18' with the module publication date
        // The format is (year-month-day)
        revision "2010-05-18" {
          description
            "Initial version";
        }

        // extension statements

        // feature statements

        // identity statements

        // typedef statements

        // grouping statements

        // data definition statements

        // augment statements

        // rpc statements

        // notification statements

        // DO NOT put deviation statements in a published module

      }


    <CODE ENDS>
```

Author’s Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2015

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems

March 04, 2015

# A YANG Data Model for Routing Management
# draft-ietf-netmod-routing-cfg-17

## Abstract

This document contains a specification of three YANG modules. Together they form the core routing data model which serves as a framework for configuring and managing a routing subsystem. It is expected that these modules will be augmented by additional YANG modules defining data models for routing protocols and other functions. The core routing data model provides common building blocks for such extensions - routing instances, routes, routing information bases (RIB), and routing protocols.

## Status of This Memo



## Copyright Notice

Table of Contents

## 1. Introduction

This document contains a specification of the following YANG modules:

- o Module "ietf-routing" provides generic components of a routing data model.

- o Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.

- o Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast, including the router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is intended as a basis for future data model development covering more sophisticated routing systems. While these three modules can be directly used for simple IP devices with static routing (see Appendix B), their main purpose is to provide essential building blocks for more complicated data models involving multiple routing protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

## 2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- o client
- o message
- o protocol operation
- o server

The following terms are defined in [RFC6020]:

- o augment
- o configuration data
- o data model
- o data node
- o feature
- o mandatory node
- o module
- o state data
- o RPC operation

## 2.1. Glossary of New Terms

core routing data model: YANG data model comprising "ietf-routing", "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing" modules.

direct route: a route to a directly connected network.

routing information base (RIB): An object containing a list of routes together with other information. See Section 5.3 for details.

system-controlled entry: An entry of a list in state data ("config false") that is created by the system independently of what has been explicitly configured. See Section 4.1 for details.

user-controlled entry: An entry of a list in state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

## 2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text.

The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- o Abbreviations before data node names: "rw" means configuration (read-write), "ro" state data (read-only), "-x" RPC operations, and "-n" notifications.
- o Symbols after data node names: "?" means an optional node, "!" a container with presence, and "*" denotes a "list" or "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2.3. Prefixes in Data Node Names

In this document, names of data nodes, RPC operations and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

| Prefix | YANG module | Reference |
|---|---|---|
| if | ietf-interfaces | [RFC7223] |
| ip | ietf-ip | [RFC7277] |
| rt | ietf-routing | Section 7 |
| v4ur | ietf-ipv4-unicast-routing | Section 8 |
| v6ur | ietf-ipv6-unicast-routing | Section 9 |
| yang | ietf-yang-types | [RFC6991] |
| inet | ietf-inet-types | [RFC6991] |

Table 1: Prefixes and corresponding YANG modules

## 3. Objectives

The initial design of the core routing data model was driven by the following objectives:

- o The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).

- o Simple routing set-ups, such as static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.

- o On the other hand, the core routing framework must allow for complicated set-ups involving multiple routing information bases (RIB) and multiple routing protocols, as well as controlled redistributions of routing information.

- o Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

## 4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Figures 1 and 2 show abridged views of the configuration and state data hierarchies. See Appendix A for the complete data trees.

```
+--rw routing
   +--rw routing-instance* [name]
   |  +--rw name
   |  +--rw type?
   |  +--rw enabled?
   |  +--rw router-id?
   |  +--rw description?
   |  +--rw default-ribs
   |  |  +--rw default-rib* [address-family]
   |  |     +--rw address-family
   |  |     +--rw rib-name
   |  +--rw interfaces
   |  |  +--rw interface* [name]
   |  |     +--rw name
   |  |     +--rw v6ur:ipv6-router-advertisements
   |  |           ...
   |  +--rw routing-protocols
   |     +--rw routing-protocol* [type name]
   |        +--rw type
   |        +--rw name
   |        +--rw description?
   |        +--rw enabled?
   |        +--rw route-preference?
   |        +--rw connected-ribs
   |        |     ...
   |        +--rw static-routes
   |              ...
   +--rw ribs
      +--rw rib* [name]
         +--rw name
         +--rw address-family
         +--rw description?
         +--rw recipient-ribs
            +--rw recipient-rib* [rib-name]
                  ...
```

Figure 1: Configuration data hierarchy.

```
+--ro routing-state
   +--ro routing-instance* [name]
   |  +--ro name
   |  +--ro type?
   |  +--ro default-ribs
   |  |  +--ro default-rib* [address-family]
   |  |     +--ro address-family
   |  |     +--ro rib-name
   |  +--ro interfaces
   |  |  +--ro interface* [name]
   |  |     +--ro name
   |  |     +--ro v6ur:ipv6-router-advertisements
   |  |           ...
   |  +--ro routing-protocols
   |     +--ro routing-protocol* [type name]
   |        +--ro type
   |        +--ro name
   |        +--ro route-preference
   |        +--ro connected-ribs
   |              ...
   +--ro ribs
      +--ro rib* [name]
         +--ro name
         +--ro address-family
         +--ro routes
         |  +--ro route*
         |        ...
         +--ro recipient-ribs
            +--ro recipient-rib* [rib-name]
                  ...
```

Figure 2: State data hierarchy.

As can be seen from Figures 1 and 2, the core routing data model introduces several generic components of a routing framework: routing instances, RIBs containing lists of routes, and routing protocols. The following subsections describe these components in more detail.

By combining the components in various ways, and possibly augmenting them with appropriate contents defined in other modules, various routing systems can be realized.

```
+--------+
| direct |       +--------------+       +--------------+
| routes |------>|              |<------|              |
+--------+       |   default    |       |  additional  |
                 |     RIB      |       |     RIB      |
+--------+       |              |       |              |
| static |------>|              |------>|              |
| routes |       +--------------+       +--------------+
+--------+           ^      |               ^      |
                     |      |               |      |
                     |      v               |      v
                  +----------+           +----------+
                  | routing  |           | routing  |
                  | protocol |           | protocol |
                  +----------+           +----------+
```

Figure 3: Example set-up of a routing system

The example in Figure 3 shows a typical (though certainly not the only possible) organization of a more complex routing subsystem for a single address family. Several of its features are worth mentioning:

- o Along with the default RIB, which is always present, an additional RIB is configured.

- o Each routing protocol instance, including the "static" and "direct" pseudo-protocols, is connected to one or more RIBs with which it can exchange routes (in both directions, except for the "static" and "direct" pseudo-protocols).

- o RIBs may also be connected to each other and exchange routes in either direction (or both).

## 4.1. System-Controlled and User-Controlled List Entries

The core routing data model defines several lists, for example "routing-instance" or "rib", that have to be populated with at least one entry in any properly functioning device, and additional entries may be configured by the user.

In such a list, the server creates the required item as a so-called system-controlled entry in state data, i.e., inside the "routing-state" container.

Additional entries may be created in the configuration by the user, e.g., via the NETCONF protocol. These are so-called user-controlled entries. If the server accepts a configured user-controlled entry, then this entry also appears in the state data version of the list.

Corresponding entries in both versions of the list (in state data and configuration) have the same value of the list key.

The user may also provide supplemental configuration of system-controlled entries. To do so, the user creates a new entry in the configuration with the desired contents. In order to bind this entry with the corresponding entry in the state data list, the key of the configuration entry has to be set to the same value as the key of the state entry.

An example can be seen in Appendix D: the "/routing-state/routing-instance" list has a single system-controlled entry whose "name" key has the value "rtr0". This entry is configured by the "/routing/routing-instance" entry whose "name" key is also "rtr0".

Deleting a user-controlled entry from the configuration list results in the removal of the corresponding entry in the state data list. In contrast, if a system-controlled entry is deleted from the configuration list, only the extra configuration specified in that entry is removed but the corresponding state data entry remains in the list.

## 5. Basic Building Blocks

This section describes the essential components of the core routing data model.

### 5.1. Routing Instance

The core routing data model supports one or more routing instances appearing as entries of the "routing-instance" list. Each routing instance has separate configuration and state data under "/rt:routing/rt:routing-instance" and "/rt:routing-state/rt:routing-instance", respectively.

The semantics of the term "routing instance" is deliberately left undefined. It is expected that future YANG modules will define data models for specific types of routing instances, such as VRF (virtual routing and forwarding) instances that are used for BGP/MPLS virtual private networks [RFC4364]. For each type of routing instance, an identity derived from "rt:routing-instance" MUST be defined. This identity is then referred to by the value of the "type" leaf (a child node of "routing-instance" list).

An implementation MAY create one or more system-controlled routing instances, and MAY also impose restrictions on types of routing instances that can be configured, and on the maximum number of supported instances for each type. For example, a simple router

implementation may support only one system-controlled routing instance of the default type "rt:default-routing-instance" and may not allow creation of any user-controlled instances.

Each network layer interface has to be assigned to one or more routing instances in order to be able to participate in packet forwarding, routing protocols and other operations of those routing instances. The assignment is accomplished by placing a corresponding (system- or user-controlled) entry in the list of routing instance interfaces ("rt:interface"). The key of the list entry is the name of a configured network layer interface, see the "ietf-interfaces" module [RFC7223].

A data model for a routing instance type MAY state additional rules for the assignment of interfaces to routing instances of that type. For example, it may be required that the sets of interfaces assigned to different routing instances of a certain type be disjoint.

#### 5.1.1. Parameters of IPv6 Routing Instance Interfaces

The module "ietf-ipv6-unicast-routing" augments the definition of the data node "rt:interface", in both configuration and state data, with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

- o send-advertisements,
- o max-rtr-adv-interval,
- o min-rtr-adv-interval,
- o managed-flag,
- o other-config-flag,
- o link-mtu,
- o reachable-time,
- o retrans-timer,
- o cur-hop-limit,
- o default-lifetime,
- o prefix-list: a list of prefixes to be advertised.

The following parameters are associated with each prefix in the list:

* valid-lifetime,

* on-link-flag,

* preferred-lifetime,

* autonomous-flag.

The definitions and descriptions of the above parameters can be found in the module "ietf-ipv6-unicast-routing" (Section 9).

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [RFC7277] (leaf "ip:forwarding").

2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-unicast-routing" module therefore assumes the former behavior with constant values.

## 5.2. Route

Routes are basic elements of information in a routing system. The core routing data model defines only the following minimal set of route attributes:

o "destination-prefix": IP prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.

o "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among routes with the same destination prefix. A lower value means a more preferred route.

o "next-hop": determines the action to be performed with a packet. See below for details.

The choice of next-hops comprises the following cases:

- o simple next-hop - IP address of the next-hop router, outgoing interface, or both.

- o special next-hop - a keyword indicating special packet handling, one of:

  - * "blackhole" - silently discard the packet;

  - * "unreachable" - discard the packet and notify the sender with a "destination unreachable" error message;

  - * "prohibit" - discard the packet notify the sender with an "administratively prohibited" error message.

It is expected that future YANG modules defining will augment routes with more complex next-hop types, or additional attributes such as metrics.

Routes are primarily state data that appear as entries of RIBs (Section 5.3) but they may also be found in configuration data, for example as manually configured static routes. In the latter case, configurable route attributes are generally a subset of route attributes described above.

## 5.3. Routing Information Base (RIB)

A routing information base (RIB) is a list of routes complemented with administrative data, namely:

- o "source-protocol": type of the routing protocol from which the route was originally obtained.

- o "active": an implementation can use this empty leaf to indicate that the route is preferred among all routes in the same RIB that have the same destination prefix.

- o "last-updated": the date and time when the route was last updated, or inserted into the RIB.

Each RIB MUST contain only routes of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing-state/ribs/rib". The contents of RIBs are controlled and manipulated by routing protocol operations which

may result in route additions, removals and modifications. This also includes manipulations via the "static" and/or "direct" pseudo-protocols, see Section 5.4.1.

RIBs are global, which means that a RIB may be used by any or all routing instances. However, a data model for a routing instance type MAY state rules and restrictions for sharing RIBs among routing instances of that type.

Each routing instance has, for every supported address family, one RIB selected as the so-called default RIB. This selection is recorded in the list "default-rib". The role of default RIBs is explained in Section 5.4.

Simple router implementations that do not advertise the feature "multiple-ribs" will typically create one system-controlled RIB per supported address family, and declare it as the default RIB (via a system-controlled entry of the "default-rib" list).

### 5.3.1. Multiple RIBs per Address Family

More complex router implementations advertising the "multiple-ribs" feature support multiple RIBs per address family that can be used for policy routing and other purposes. Every RIB can then serve as a source of routes for other RIBs of the same address family. To achieve this, one or more recipient RIBs may be specified in the configuration of the source RIB.

A RIB MUST NOT appear among its own recipient RIBs.

## 5.4. Routing Protocol

The core routing data model provides an open-ended framework for defining multiple routing protocol instances within a routing instance. Each routing protocol instance MUST be assigned a type, which is an identity derived from the "rt:routing-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.4.1).

Multiple routing protocol instances of the same type MAY be configured within the same routing instance.

Each routing protocol instance can be connected to one or more RIBs for each address family that the routing protocol instance supports. By default, the interaction of a routing protocol instance with its connected RIBs is governed by the following rules:

- o Routes learned from the network are installed in all connected RIBs with a matching address family.

- o Conversely, routes from all connected RIBs are injected into the routing protocol instance.

However, a data model for a routing protocol MAY impose specific rules for exchanging routes between routing protocol instances and connected RIBs.

On devices supporting the "multiple-ribs" feature, any RIB (system-controlled or user-controlled) may be connected to a routing protocol instance by configuring a corresponding entry in the "connected-rib" list. If such an entry is not configured for an address family, then the default RIB MUST be used as the connected RIB for this address family.

### 5.4.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types - "direct" and "static". Both are in fact pseudo-protocols, which means they are confined to the local device and do not exchange any routing information with adjacent routers. Routes from both "direct" and "static" protocol instances are passed to the connected RIBs, but an exchange in the opposite direction is not allowed.

Every routing instance MUST implement exactly one instance of the "direct" pseudo-protocol type. It is the source of direct routes for all configured address families. Direct routes are normally supplied by the operating system kernel, based on the configuration of network interface addresses, see Section 6.2. The "direct" pseudo-protocol MUST always be connected to the default RIBs of all supported address families. Unlike other routing protocol types, this connection cannot be changed in the configuration.

A pseudo-protocol of the type "static" allows for specifying routes manually. It MAY be configured in zero or multiple instances, although a typical configuration will have exactly one instance per routing instance.

### 5.4.2. Defining New Routing Protocols

It is expected that future YANG modules will create data models for additional routing protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to fit it into the core routing framework in the following way:

- o A new identity MUST be defined for the routing protocol and its base identity MUST be set to "rt:routing-protocol", or to an identity derived from "rt:routing-protocol".

- o Additional route attributes MAY be defined, preferably in one place by means of defining a YANG grouping. The new attributes have to be inserted by augmenting the definitions of the nodes

  /rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route

  and

  /rt:fib-route/rt:output/rt:route,

  and possibly other places in the configuration, state data, notifications, and RPC input or output.

- o Configuration parameters and/or state data for the new protocol can be defined by augmenting the "routing-protocol" data node under both "/routing" and "/routing-state".

- o Per-interface configuration, including activation of the routing protocol on individual interfaces, can use references to entries in the list of routing instance interfaces (rt:interface).

By using the "when" statement, the augmented configuration parameters and state data specific to the new protocol SHOULD be made conditional and valid only if the value of "rt:type" or "rt:source-protocol" is equal to the new protocol’s identity. It is also RECOMMENDED that protocol-specific data nodes be encapsulated in appropriately named containers.

The above steps are implemented by the example YANG module for the RIP routing protocol in Appendix C.

## 5.5. RPC Operations

The "ietf-routing" module defines two RPC operations:

- o fib-route: query a routing instance for the active route in the Forwarding Information Base (FIB). It is the route that is currently used for sending datagrams to a destination host whose address is passed as an input parameter.

- o route-count: retrieve the total number of entries in a RIB.

## 6. Interactions with Other YANG Modules

The semantics of the core routing data model also depends on several configuration parameters that are defined in other YANG modules.

### 6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [RFC7223]:

/if:interfaces/if:interface/if:enabled

If this switch is set to "false" for a network layer interface, the device MUST behave exactly as if that interface was not assigned to any routing instance at all.

### 6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [RFC7277]:

/if:interfaces/if:interface/ip:ipv4/ip:enabled

If this switch is set to "false" for a network layer interface, then all IPv4 routing functions related to that interface MUST be disabled.

/if:interfaces/if:interface/ip:ipv4/ip:forwarding

If this switch is set to "false" for a network layer interface, then the forwarding of IPv4 datagrams to and from this interface MUST be disabled. However, the interface may participate in other IPv4 routing functions, such as routing protocols.

/if:interfaces/if:interface/ip:ipv6/ip:enabled

If this switch is set to "false" for a network layer interface, then all IPv6 routing functions related to that interface MUST be disabled.

/if:interfaces/if:interface/ip:ipv6/ip:forwarding

If this switch is set to "false" for a network layer interface, then the forwarding of IPv6 datagrams to and from this interface MUST be disabled. However, the interface may participate in other IPv6 routing functions, such as routing protocols.

In addition, the "ietf-ip" module allows for configuring IPv4 and IPv6 addresses and network prefixes or masks on network layer interfaces. Configuration of these parameters on an enabled interface MUST result in an immediate creation of the corresponding direct route. The destination prefix of this route is set according to the configured IP address and network prefix/mask, and the interface is set as the outgoing interface for that route.

## 7. Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-routing@2015-03-04.yang"

module ietf-routing {

  namespace "urn:ietf:params:xml:ns:yang:ietf-routing";

  prefix "rt";

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-interfaces {
    prefix "if";
  }

  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: Thomas Nadeau
               <mailto:tnadeau@lucidvision.com>

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

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>";

  description
    "This YANG module defines essential components for the management
```

```
        of a routing subsystem.

        Copyright (c) 2014 IETF Trust and the persons identified as
        authors of the code. All rights reserved.

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

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

     revision 2015-03-04 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* Features */

     feature multiple-ribs {
       description
         "This feature indicates that the server supports user-defined
          RIBS and the framework for passing routes between RIBs.

          Servers that do not advertise this feature MUST provide
          exactly one system-controlled RIB per supported address family
          and make them also the default RIBs. These RIBs then appear as
          entries of the list /routing-state/ribs/rib.";
     }

     feature router-id {
       description
         "This feature indicates that the server supports configuration
          of an explicit 32-bit router ID that is used by some routing
          protocols.

          Servers that do not advertise this feature set a router ID
          algorithmically, usually to one of configured IPv4 addresses.
          However, this algorithm is implementation-specific.";
     }

     /* Identities */
```

```
identity address-family {
  description
    "Base identity from which identities describing address
     families are derived.";
}

identity ipv4 {
  base address-family;
  description
    "This identity represents IPv4 address family.";
}

identity ipv6 {
  base address-family;
  description
    "This identity represents IPv6 address family.";
}

identity routing-instance {
  description
    "Base identity from which identities describing routing
     instance types are derived.";
}

identity default-routing-instance {
  base routing-instance;
  description
    "This identity represents either a default routing instance, or
     the only routing instance on systems that do not support
     multiple instances.";
}

identity routing-protocol {
  description
    "Base identity from which routing protocol identities are
     derived.";
}

identity direct {
  base routing-protocol;
  description
    "Routing pseudo-protocol that provides routes to directly
     connected networks.";
}

identity static {
  base routing-protocol;
  description
```

```
         "Static routing pseudo-protocol.";
     }

     /* Type Definitions */

     typedef routing-instance-ref {
       type leafref {
         path "/rt:routing/rt:routing-instance/rt:name";
       }
       description
         "This type is used for leafs that reference a routing instance
          configuration.";
     }

     typedef routing-instance-state-ref {
       type leafref {
         path "/rt:routing-state/rt:routing-instance/rt:name";
       }
       description
         "This type is used for leafs that reference state data of a
          routing instance.";
     }

     typedef rib-ref {
       type leafref {
         path "/rt:routing/rt:ribs/rt:rib/rt:name";
       }
       description
         "This type is used for leafs that reference a RIB
          configuration.";
     }

     typedef rib-state-ref {
       type leafref {
         path "/rt:routing-state/rt:ribs/rt:rib/rt:name";
       }
       description
         "This type is used for leafs that reference a RIB in state
          data.";
     }

     typedef route-preference {
       type uint32;
       description
         "This type is used for route preferences.";
     }

     /* Groupings */
```

```
     grouping address-family {
       description
         "This grouping provides a leaf identifying an address
          family.";
       leaf address-family {
         type identityref {
           base address-family;
         }
         mandatory "true";
         description
           "Address family.";
       }
     }

     grouping router-id {
       description
         "This grouping provides router ID.";
       leaf router-id {
         type yang:dotted-quad;
         description
           "A 32-bit number in the form of a dotted quad that is used by
            some routing protocols identifying a router.";
         reference
           "RFC 2328: OSPF Version 2.";
       }
     }

     grouping special-next-hop {
       description
         "This grouping provides a leaf with an enumeration of special
          next-hops.";
       leaf special-next-hop {
         type enumeration {
           enum blackhole {
             description
               "Silently discard the packet.";
           }
           enum unreachable {
             description
               "Discard the packet and notify the sender with an error
                message indicating that the destination host is
                unreachable.";
           }
           enum prohibit {
             description
               "Discard the packet and notify the sender with an error
                message indicating that the communication is
                administratively prohibited.";
```

```
          }
          enum receive {
            description
              "The packet will be received by the local system.";
          }
        }
        description
          "Special next-hop options.";
      }
    }

    grouping next-hop-content {
      description
        "Generic parameters of next-hops in static routes.";
      choice next-hop-options {
        mandatory "true";
        description
          "Options for next-hops in static routes.

           It is expected that other cases will be added through
           augments from other modules, e.g., for ECMP.";
        case simple-next-hop {
          description
            "Simple next-hop is specified as an outgoing interface,
             next-hop address or both.

             Address-family-specific modules are expected to provide
             'next-hop-address' leaf via augmentation.";
          leaf outgoing-interface {
            type leafref {
              path "/rt:routing/rt:routing-instance/rt:interfaces/"
                 + "rt:interface/rt:name";
            }
            description
              "Name of the outgoing interface.";
          }
        }
        case special-next-hop {
          uses special-next-hop;
        }
      }
    }

    grouping next-hop-state-content {
      description
        "Generic parameters of next-hops in state data.";
      choice next-hop-options {
        mandatory "true";
```

```
        description
          "Options for next-hops in state data.

           It is expected that other cases will be added through
           augments from other modules, e.g., for ECMP or recursive
           next-hops.";
        case simple-next-hop {
          description
            "Simple next-hop is specified as an outgoing interface,
             next-hop address or both.

             Address-family-specific modules are expected to provide
             'next-hop-address' leaf via augmentation.";
          leaf outgoing-interface {
            type leafref {
              path "/rt:routing-state/rt:routing-instance/"
                 + "rt:interfaces/rt:interface/rt:name";
            }
            description
              "Name of the outgoing interface.";
          }
        }
        case special-next-hop {
          uses special-next-hop;
        }
      }
    }

    grouping route-metadata {
      description
        "Common route metadata.";
      leaf source-protocol {
        type identityref {
          base routing-protocol;
        }
        mandatory "true";
        description
          "Type of the routing protocol from which the route
           originated.";
      }
      leaf active {
        type empty;
        description
          "Presence of this leaf indicates that the route is preferred
           among all routes in the same RIB that have the same
           destination prefix.";
      }
      leaf last-updated {
```

```
        type yang:date-and-time;
        description
          "Time stamp of the last modification of the route. If the
           route was never modified, it is the time when the route was
           inserted into the RIB.";
      }
    }

    /* State data */

    augment "/if:interfaces-state/if:interface" {
      description
        "This augment adds a wrapped leaf-list to interface state
         data.";
      container routing-instances {
        description
          "The enclosed leaf-list provides references to all routing
           instances to which the parent interface is assigned.

           The assignments are configured as a part of routing-instance
           configuration (module ietf-routing).";
        leaf-list routing-instance {
          type routing-instance-state-ref;
          must "../../if:name=/rt:routing-state/"
             + "rt:routing-instance[rt:name=current()]/rt:interfaces/"
             + "rt:interface/rt:name" {
            error-message "The interface is not assigned to the "
                        + "routing instance.";
            description
              "The reference must mirror a corresponding assignment
               under routing-instance.";
          }
          description
            "Reference to a routing instance.";
        }
      }
    }

    container routing-state {
      config "false";
      description
        "State data of the routing subsystem.";
      list routing-instance {
        key "name";
        min-elements "1";
        description
          "Each list entry is a container for state data of a routing
           instance.
```

```
            An implementation MAY create one or more system-controlled
            instances, other user-controlled instances MAY be created by
            configuration.";
        leaf name {
          type string;
          description
            "The name of the routing instance.

             For system-controlled instances the name is persistent,
             i.e., it SHOULD NOT change across reboots.";
        }
        leaf type {
          type identityref {
            base routing-instance;
          }
          description
            "The routing instance type.";
        }
        container default-ribs {
          description
            "Default RIBs used by the routing instance.";
          list default-rib {
            key "address-family";
            description
              "Each list entry specifies the default RIB for one
               address family.

               The default RIB is operationally connected to all
               routing protocols for which a connected RIB has not been
               explicitly configured.

               The 'direct' pseudo-protocol is always connected to the
               default RIBs.";
            uses address-family;
            leaf rib-name {
              type rib-state-ref;
              mandatory "true";
              description
                "Name of an existing RIB to be used as the default RIB
                 for the given routing instance and address family.";
            }
          }
        }
        container interfaces {
          description
            "Network layer interfaces belonging to the routing
             instance.";
          list interface {
```

```
            key "name";
            description
              "List of network layer interfaces assigned to the routing
               instance.";
            leaf name {
              type if:interface-state-ref;
              description
                "A reference to the name of a configured network layer
                 interface.";
            }
          }
        }
        container routing-protocols {
          description
            "Container for the list of routing protocol instances.";
          list routing-protocol {
            key "type name";
            description
              "State data of a routing protocol instance.

               An implementation MUST provide exactly one
               system-controlled instance of the type 'direct'. Other
               instances MAY be created by configuration.";
            leaf type {
              type identityref {
                base routing-protocol;
              }
              description
                "Type of the routing protocol.";
            }
            leaf name {
              type string;
              description
                "The name of the routing protocol instance.

                 For system-controlled instances this name is
                 persistent, i.e., it SHOULD NOT change across
                 reboots.";
            }
            leaf route-preference {
              type route-preference;
              mandatory "true";
              description
                "The value of route preference (administrative
                 distance) assigned to all routes generated by the
                 routing protocol instance. A lower value means a more
                 preferred route.";
            }
```

```
            container connected-ribs {
              description
                "Container for connected RIBs.";
              list connected-rib {
                key "rib-name";
                description
                  "List of RIBs to which the routing protocol instance
                   is connected.

                   By default, routes learned by the routing protocol
                   instance are installed in all connected RIBs of the
                   matching address family, and, conversely, all routes
                   from connected RIBs are installed in the routing
                   protocol instance. However, routing protocols may
                   specify other rules.";
                leaf rib-name {
                  type rib-state-ref;
                  description
                    "Name of an existing RIB.";
                }
              }
            }
          }
        }
      }
      container ribs {
        description
          "Container for RIBs.";
        list rib {
          key "name";
          description
            "Each entry represents a RIB identified by the 'name' key.
             All routes in a RIB MUST belong to the same address
             family.

             The server MUST provide a system-controlled default RIB
             for each supported address family, and MAY provide other
             system-controlled RIBs. Additional RIBs MAY be created in
             the configuration.";
          leaf name {
            type string;
            description
              "The name of the RIB.";
          }
          uses address-family;
          container routes {
            description
              "Current content of the RIB.";
```

```
            list route {
              description
                "A RIB route entry. This data node MUST be augmented
                 with information specific for routes of each address
                 family.";
              leaf route-preference {
                type route-preference;
                description
                  "This route attribute, also known as administrative
                   distance, allows for selecting the preferred route
                   among routes with the same destination prefix. A
                   smaller value means a more preferred route.";
              }
              container next-hop {
                description
                  "Route's next-hop attribute.";
                uses next-hop-state-content;
              }
              uses route-metadata;
            }
          }
          container recipient-ribs {
            description
              "Container for recipient RIBs.";
            list recipient-rib {
              key "rib-name";
              description
                "List of RIBs that receive routes from this RIB.";
              leaf rib-name {
                type rib-state-ref;
                description
                  "The name of the recipient RIB.";
              }
            }
          }
        }
      }
    }

    /* Configuration Data */

    container routing {
      description
        "Configuration parameters for the routing subsystem.";
      list routing-instance {
        key "name";
        description
          "Configuration of a routing instance.";
```

```
          leaf name {
            type string;
            description
              "The name of the routing instance.

               For system-controlled entries, the value of this leaf must
               be the same as the name of the corresponding entry in
               state data.

               For user-controlled entries, an arbitrary name can be
               used.";
          }
          leaf type {
            type identityref {
              base routing-instance;
            }
            default "rt:default-routing-instance";
            description
              "The type of the routing instance.";
          }
          leaf enabled {
            type boolean;
            default "true";
            description
              "Enable/disable the routing instance.

               If this parameter is false, the parent routing instance is
               disabled and does not appear in state data, despite any
               other configuration that might be present.";
          }
          uses router-id {
            if-feature router-id;
            description
              "Configuration of the global router ID. Routing protocols
               that use router ID can use this parameter or override it
               with another value.";
          }
          leaf description {
            type string;
            description
              "Textual description of the routing instance.";
          }
          container default-ribs {
            if-feature multiple-ribs;
            description
              "Configuration of the default RIBs used by the routing
               instance.
```

```
              The default RIB for an addressed family if by default
              connected to all routing protocol instances supporting
              that address family, and always receives direct routes.";
            list default-rib {
              must "address-family=/routing/ribs/rib[name=current()/"
                 + "rib-name]/address-family" {
                error-message "Address family mismatch.";
                description
                  "The entry's address family MUST match that of the
                   referenced RIB.";
              }
              key "address-family";
              description
                "Each list entry configures the default RIB for one
                 address family.";
              uses address-family;
              leaf rib-name {
                type string;
                mandatory "true";
                description
                  "Name of an existing RIB to be used as the default RIB
                   for the given routing instance and address family.";
              }
            }
          }
          container interfaces {
            description
              "Configuration of the routing instance's interfaces.";
            list interface {
              key "name";
              description
                "List of network layer interfaces assigned to the routing
                 instance.";
              leaf name {
                type if:interface-ref;
                description
                  "A reference to the name of a configured network layer
                   interface.";
              }
            }
          }
          container routing-protocols {
            description
              "Configuration of routing protocol instances.";
            list routing-protocol {
              key "type name";
              description
                "Each entry contains configuration of a routing protocol
```

```
               instance.";
            leaf type {
              type identityref {
                base routing-protocol;
              }
              description
                "Type of the routing protocol - an identity derived
                 from the 'routing-protocol' base identity.";
            }
            leaf name {
              type string;
              description
                "An arbitrary name of the routing protocol instance.";
            }
            leaf description {
              type string;
              description
                "Textual description of the routing protocol
                 instance.";
            }
            leaf enabled {
              type boolean;
              default "true";
              description
                "Enable/disable the routing protocol instance.

                 If this parameter is false, the parent routing
                 protocol instance is disabled and does not appear in
                 state data, despite any other configuration that might
                 be present.";
            }
            leaf route-preference {
              type route-preference;
              description
                "The value of route preference (administrative
                 distance).

                 The default value depends on the routing protocol
                 type, and may also be implementation-dependent.";
            }
            container connected-ribs {
              description
                "Configuration of connected RIBs.";
              list connected-rib {
                key "rib-name";
                description
                  "Each entry configures a RIB to which the routing
                   protocol instance is connected.
```

```
                    If no connected RIB is configured for an address
                    family, the routing protocol is connected to the
                    default RIB for that address family.";
                  leaf rib-name {
                    type rib-ref;
                    must "../../../type != 'rt:direct' or "
                       + "../../../../../default-ribs/ "
                       + "default-rib/rib-name=." {
                      error-message "The 'direct' protocol can be "
                                  + "connected only to a default RIB.";
                      description
                        "For the 'direct' pseudo-protocol, the connected
                         RIB must always be a default RIB.";
                    }
                    description
                      "Name of an existing RIB.";
                  }
                }
              }
              container static-routes {
                when "../type='rt:static'" {
                  description
                    "This container is only valid for the 'static'
                     routing protocol.";
                }
                description
                  "Configuration of the 'static' pseudo-protocol.

                   Address-family-specific modules augment this node with
                   their lists of routes.";
              }
            }
          }
        }
        container ribs {
          description
            "Configuration of RIBs.";
          list rib {
            key "name";
            description
              "Each entry represents a configured RIB identified by the
               'name' key.

               Entries having the same key as a system-controlled entry
               of the list /routing-state/ribs/rib are used for
               configuring parameters of that entry. Other entries define
               additional user-controlled RIBs.";
            leaf name {
```

```
            type string;
            description
              "The name of the RIB.

               For system-controlled entries, the value of this leaf
               must be the same as the name of the corresponding entry
               in state data.

               For user-controlled entries, an arbitrary name can be
               used.";
          }
          uses address-family;
          leaf description {
            type string;
            description
              "Textual description of the RIB.";
          }
          container recipient-ribs {
            if-feature multiple-ribs;
            description
              "Configuration of recipient RIBs.";
            list recipient-rib {
              must "rib-name != ../../name" {
                error-message
                  "Source and recipient RIBs are identical.";
                description
                  "A RIB MUST NOT appear among its recipient RIBs.";
              }
              must "/routing/ribs/rib[name=current()/rib-name]/"
                 + "address-family=../../address-family" {
                error-message "Address family mismatch.";
                description
                  "Address family of the recipient RIB MUST match that
                   of the source RIB.";
              }
              key "rib-name";
              description
                "Each entry configures a recipient RIB.";
              leaf rib-name {
                type rib-ref;
                description
                  "The name of the recipient RIB.";
              }
            }
          }
        }
      }
    }
```

```
/* RPC operations */

rpc fib-route {
  description
    "Return the active FIB route that a routing-instance uses for
     sending packets to a destination address.";
  input {
    leaf routing-instance-name {
      type routing-instance-state-ref;
      mandatory "true";
      description
        "Name of the routing instance whose forwarding information
         base is being queried.

         If the routing instance with name equal to the value of
         this parameter doesn't exist, then this operation SHALL
         fail with error-tag 'data-missing' and error-app-tag
         'routing-instance-not-found'.";
    }
    container destination-address {
      description
        "Network layer destination address.

         Address family specific modules MUST augment this
         container with a leaf named 'address'.";
      uses address-family;
    }
  }
  output {
    container route {
      description
        "The active FIB route for the specified destination.

         If the routing instance has no active FIB route for the
         destination address, no output is returned - the server
         SHALL send an <rpc-reply> containing a single element
         <ok>.

         Address family specific modules MUST augment this list
         with appropriate route contents.";
      uses address-family;
      container next-hop {
        description
          "Route's next-hop attribute.";
        uses next-hop-state-content;
      }
      uses route-metadata;
    }
```

```
      }
    }

    rpc route-count {
      description
        "Return the current number of routes in a RIB.";
      input {
        leaf rib-name {
          type rib-state-ref;
          mandatory "true";
          description
            "Name of the RIB.

             If the RIB with name equal to the value of this parameter
             doesn't exist, then this operation SHALL fail with
             error-tag 'data-missing' and error-app-tag
             'rib-not-found'.";
        }
      }
      output {
        leaf number-of-routes {
          type uint64;
          mandatory "true";
          description
            "Number of routes in the RIB.";
        }
      }
    }
  }

  <CODE ENDS>
```

## 8. IPv4 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
  <CODE BEGINS> file "ietf-ipv4-unicast-routing@2015-02-10.yang"

  module ietf-ipv4-unicast-routing {

    namespace "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing";

    prefix "v4ur";

    import ietf-routing {
      prefix "rt";
```

```
     }

     import ietf-inet-types {
       prefix "inet";
     }

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

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

        WG Chair: Thomas Nadeau
                  <mailto:tnadeau@lucidvision.com>

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

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>";

     description
       "This YANG module augments the 'ietf-routing' module with basic
        configuration and state data for IPv4 unicast routing.

        Copyright (c) 2014 IETF Trust and the persons identified as
        authors of the code. All rights reserved.

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

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

     revision 2015-02-10 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* Identities */
```

```
identity ipv4-unicast {
  base rt:ipv4;
  description
    "This identity represents the IPv4 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "../../rt:address-family = 'v4ur:ipv4-unicast'" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "This leaf augments an IPv4 unicast route.";
  leaf destination-prefix {
    type inet:ipv4-prefix;
    description
      "IPv4 destination prefix.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
      + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
  when "../../../rt:address-family = 'v4ur:ipv4-unicast'" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "This leaf augments the 'simple-next-hop' case of IPv4 unicast
     routes.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

/* Configuration data */

augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
      + "rt:routing-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
     pseudo-protocol with data specific to IPv4 unicast.";
  container ipv4 {
    description
      "Configuration of a 'static' pseudo-protocol instance
```

```
           consists of a list of routes.";
         list route {
           key "destination-prefix";
           ordered-by "user";
           description
             "A user-ordered list of static routes.";
           leaf destination-prefix {
             type inet:ipv4-prefix;
             mandatory "true";
             description
               "IPv4 destination prefix.";
           }
           leaf description {
             type string;
             description
               "Textual description of the route.";
           }
           container next-hop {
             description
               "Configuration of next-hop.";
             uses rt:next-hop-content {
               augment "next-hop-options" {
                 description
                   "Add next-hop address case.";
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next-hop.";
                 }
               }
             }
           }
         }
       }
     }

     /* RPC operations */

     augment "/rt:fib-route/rt:input/rt:destination-address" {
       when "rt:address-family='v4ur:ipv4-unicast'" {
         description
           "This augment is valid only for IPv4 unicast.";
       }
       description
         "This leaf augments the 'rt:destination-address' parameter of
          the 'rt:fib-route' operation.";
       leaf address {
         type inet:ipv4-address;
```

```
        description
          "IPv4 destination address.";
      }
    }

    augment "/rt:fib-route/rt:output/rt:route" {
      when "rt:address-family='v4ur:ipv4-unicast'" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "This leaf augments the reply to the 'rt:fib-route'
         operation.";
      leaf destination-prefix {
        type inet:ipv4-prefix;
        description
          "IPv4 destination prefix.";
      }
    }

    augment "/rt:fib-route/rt:output/rt:route/rt:next-hop/"
          + "rt:next-hop-options/rt:simple-next-hop" {
      when "../rt:address-family='v4ur:ipv4-unicast'" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "This leaf augments the 'simple-next-hop' case in the reply to
         the 'rt:fib-route' operation.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }
  }

  <CODE ENDS>
```

## 9. IPv6 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
  <CODE BEGINS> file "ietf-ipv6-unicast-routing@2015-02-10.yang"

  module ietf-ipv6-unicast-routing {
```

```
namespace "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";

prefix "v6ur";

import ietf-routing {
  prefix "rt";
}

import ietf-inet-types {
  prefix "inet";
}

import ietf-interfaces {
  prefix "if";
}

import ietf-ip {
  prefix "ip";
}

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: Thomas Nadeau
             <mailto:tnadeau@lucidvision.com>

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

   Editor:   Ladislav Lhotka
             <mailto:lhotka@nic.cz>";

description
  "This YANG module augments the 'ietf-routing' module with basic
   configuration and state data for IPv6 unicast routing.

   Copyright (c) 2014 IETF Trust and the persons identified as
   authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject to
   the license terms contained in, the Simplified BSD License set
   forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
```

```
        (http://trustee.ietf.org/license-info).

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

     revision 2015-02-10 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* Identities */

     identity ipv6-unicast {
       base rt:ipv6;
       description
         "This identity represents the IPv6 unicast address family.";
     }

     /* State data */

     augment "/rt:routing-state/rt:routing-instance/rt:interfaces/"
           + "rt:interface" {
       description
         "IPv6-specific parameters of router interfaces.";
       container ipv6-router-advertisements {
         description
           "Parameters of IPv6 Router Advertisements.";
         leaf send-advertisements {
           type boolean;
           description
             "A flag indicating whether or not the router sends periodic
              Router Advertisements and responds to Router
              Solicitations.";
         }
         leaf max-rtr-adv-interval {
           type uint16 {
             range "4..1800";
           }
           units "seconds";
           description
             "The maximum time allowed between sending unsolicited
              multicast Router Advertisements from the interface.";
         }
         leaf min-rtr-adv-interval {
           type uint16 {
             range "3..1350";
```

```
          }
          units "seconds";
          description
            "The minimum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.";
        }
        leaf managed-flag {
          type boolean;
          description
            "The value that is placed in the 'Managed address
             configuration' flag field in the Router Advertisement.";
        }
        leaf other-config-flag {
          type boolean;
          description
            "The value that is placed in the 'Other configuration' flag
             field in the Router Advertisement.";
        }
        leaf link-mtu {
          type uint32;
          description
            "The value that is placed in MTU options sent by the
             router. A value of zero indicates that no MTU options are
             sent.";
        }
        leaf reachable-time {
          type uint32 {
            range "0..3600000";
          }
          units "milliseconds";
          description
            "The value that is placed in the Reachable Time field in
             the Router Advertisement messages sent by the router. A
             value of zero means unspecified (by this router).";
        }
        leaf retrans-timer {
          type uint32;
          units "milliseconds";
          description
            "The value that is placed in the Retrans Timer field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).";
        }
        leaf cur-hop-limit {
          type uint8;
          description
            "The value that is placed in the Cur Hop Limit field in the
             Router Advertisement messages sent by the router. A value
```

```
              of zero means unspecified (by this router).";
          }
          leaf default-lifetime {
            type uint16 {
              range "0..9000";
            }
            units "seconds";
            description
              "The value that is placed in the Router Lifetime field of
               Router Advertisements sent from the interface, in seconds.
               A value of zero indicates that the router is not to be
               used as a default router.";
          }
          container prefix-list {
            description
              "A list of prefixes that are placed in Prefix Information
               options in Router Advertisement messages sent from the
               interface.

               By default, these are all prefixes that the router
               advertises via routing protocols as being on-link for the
               interface from which the advertisement is sent.";
            list prefix {
              key "prefix-spec";
              description
                "Advertised prefix entry and its parameters.";
              leaf prefix-spec {
                type inet:ipv6-prefix;
                description
                  "IPv6 address prefix.";
              }
              leaf valid-lifetime {
                type uint32;
                units "seconds";
                description
                  "The value that is placed in the Valid Lifetime in the
                   Prefix Information option. The designated value of all
                   1's (0xffffffff) represents infinity.";
              }
              leaf on-link-flag {
                type boolean;
                description
                  "The value that is placed in the on-link flag ('L-bit')
                   field in the Prefix Information option.";
              }
              leaf preferred-lifetime {
                type uint32;
                units "seconds";
```

```
              description
                "The value that is placed in the Preferred Lifetime in
                 the Prefix Information option, in seconds. The
                 designated value of all 1's (0xffffffff) represents
                 infinity.";
            }
            leaf autonomous-flag {
              type boolean;
              description
                "The value that is placed in the Autonomous Flag field
                 in the Prefix Information option.";
            }
          }
        }
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
      when "../../rt:address-family = 'v6ur:ipv6-unicast'" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This leaf augments an IPv6 unicast route.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        description
          "IPv6 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "../../../rt:address-family = 'v6ur:ipv6-unicast'" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This leaf augments the 'simple-next-hop' case of IPv6 unicast
         routes.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    /* Configuration data */
```

```
     augment
       "/rt:routing/rt:routing-instance/rt:interfaces/rt:interface" {
       when "/if:interfaces/if:interface[if:name=current()/rt:name]/"
          + "ip:ipv6/ip:enabled='true'" {
         description
           "This augment is only valid for router interfaces with
            enabled IPv6.";
       }
       description
         "Configuration of IPv6-specific parameters of router
          interfaces.";
       container ipv6-router-advertisements {
         description
           "Configuration of IPv6 Router Advertisements.";
         leaf send-advertisements {
           type boolean;
           default "false";
           description
             "A flag indicating whether or not the router sends periodic
              Router Advertisements and responds to Router
              Solicitations.";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvSendAdvertisements.";
         }
         leaf max-rtr-adv-interval {
           type uint16 {
             range "4..1800";
           }
           units "seconds";
           default "600";
           description
             "The maximum time allowed between sending unsolicited
              multicast Router Advertisements from the interface.";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              MaxRtrAdvInterval.";
         }
         leaf min-rtr-adv-interval {
           type uint16 {
             range "3..1350";
           }
           units "seconds";
           must ". <= 0.75 * ../max-rtr-adv-interval" {
             description
               "The value MUST NOT be greater than 75 % of
                'max-rtr-adv-interval'.";
           }
```

```
          description
            "The minimum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.

             The default value to be used operationally if this leaf is
             not configured is determined as follows:

             - if max-rtr-adv-interval >= 9 seconds, the default value
               is 0.33 * max-rtr-adv-interval;

             - otherwise it is 0.75 * max-rtr-adv-interval.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MinRtrAdvInterval.";
        }
        leaf managed-flag {
          type boolean;
          default "false";
          description
            "The value to be placed in the 'Managed address
             configuration' flag field in the Router Advertisement.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvManagedFlag.";
        }
        leaf other-config-flag {
          type boolean;
          default "false";
          description
            "The value to be placed in the 'Other configuration' flag
             field in the Router Advertisement.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvOtherConfigFlag.";
        }
        leaf link-mtu {
          type uint32;
          default "0";
          description
            "The value to be placed in MTU options sent by the router.
             A value of zero indicates that no MTU options are sent.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvLinkMTU.";
        }
        leaf reachable-time {
          type uint32 {
            range "0..3600000";
```

```
          }
          units "milliseconds";
          default "0";
          description
            "The value to be placed in the Reachable Time field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvReachableTime.";
        }
        leaf retrans-timer {
          type uint32;
          units "milliseconds";
          default "0";
          description
            "The value to be placed in the Retrans Timer field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvRetransTimer.";
        }
        leaf cur-hop-limit {
          type uint8;
          description
            "The value to be placed in the Cur Hop Limit field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).

             If this parameter is not configured, the device SHOULD use
             the value specified in IANA Assigned Numbers that was in
             effect at the time of implementation.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvCurHopLimit.

             IANA: IP Parameters,
             http://www.iana.org/assignments/ip-parameters";
        }
        leaf default-lifetime {
          type uint16 {
            range "0..9000";
          }
          units "seconds";
          description
            "The value to be placed in the Router Lifetime field of
             Router Advertisements sent from the interface, in seconds.
```

```
              It MUST be either zero or between max-rtr-adv-interval and
              9000 seconds. A value of zero indicates that the router is
              not to be used as a default router. These limits may be
              overridden by specific documents that describe how IPv6
              operates over different link layers.

              If this parameter is not configured, the device SHOULD use
              a value of 3 * max-rtr-adv-interval.";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvDefaultLifeTime.";
          }
          container prefix-list {
            description
              "Configuration of prefixes to be placed in Prefix
               Information options in Router Advertisement messages sent
               from the interface.

               Prefixes that are advertised by default but do not have
               their entries in the child 'prefix' list are advertised
               with the default values of all parameters.

               The link-local prefix SHOULD NOT be included in the list
               of advertised prefixes.";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
               AdvPrefixList.";
            list prefix {
              key "prefix-spec";
              description
                "Configuration of an advertised prefix entry.";
              leaf prefix-spec {
                type inet:ipv6-prefix;
                description
                  "IPv6 address prefix.";
              }
              choice control-adv-prefixes {
                default "advertise";
                description
                  "The prefix either may be explicitly removed from the
                   set of advertised prefixes, or parameters with which
                   it is advertised may be specified (default case).";
                leaf no-advertise {
                  type empty;
                  description
                    "The prefix will not be advertised.

                     This can be used for removing the prefix from the
```

```
                 default set of advertised prefixes.";
               }
               case advertise {
                 leaf valid-lifetime {
                   type uint32;
                   units "seconds";
                   default "2592000";
                   description
                     "The value to be placed in the Valid Lifetime in
                      the Prefix Information option. The designated
                      value of all 1's (0xffffffff) represents
                      infinity.";
                   reference
                     "RFC 4861: Neighbor Discovery for IP version 6
                      (IPv6) - AdvValidLifetime.";
                 }
                 leaf on-link-flag {
                   type boolean;
                   default "true";
                   description
                     "The value to be placed in the on-link flag
                      ('L-bit') field in the Prefix Information
                      option.";
                   reference
                     "RFC 4861: Neighbor Discovery for IP version 6
                      (IPv6) - AdvOnLinkFlag.";
                 }
                 leaf preferred-lifetime {
                   type uint32;
                   units "seconds";
                   must ". <= ../valid-lifetime" {
                     description
                       "This value MUST NOT be greater than
                        valid-lifetime.";
                   }
                   default "604800";
                   description
                     "The value to be placed in the Preferred Lifetime
                      in the Prefix Information option. The designated
                      value of all 1's (0xffffffff) represents
                      infinity.";
                   reference
                     "RFC 4861: Neighbor Discovery for IP version 6
                      (IPv6) - AdvPreferredLifetime.";
                 }
                 leaf autonomous-flag {
                   type boolean;
                   default "true";
```

```
                    description
                      "The value to be placed in the Autonomous Flag
                       field in the Prefix Information option.";
                    reference
                      "RFC 4861: Neighbor Discovery for IP version 6
                       (IPv6) - AdvAutonomousFlag.";
                  }
                }
              }
            }
          }
        }
      }

      augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
            + "rt:routing-protocol/rt:static-routes" {
        description
          "This augment defines the configuration of the 'static'
           pseudo-protocol with data specific to IPv6 unicast.";
        container ipv6 {
          description
            "Configuration of a 'static' pseudo-protocol instance
             consists of a list of routes.";
          list route {
            key "destination-prefix";
            ordered-by "user";
            description
              "A user-ordered list of static routes.";
            leaf destination-prefix {
              type inet:ipv6-prefix;
              mandatory "true";
              description
                "IPv6 destination prefix.";
            }
            leaf description {
              type string;
              description
                "Textual description of the route.";
            }
            container next-hop {
              description
                "Configuration of next-hop.";
              uses rt:next-hop-content {
                augment "next-hop-options" {
                  description
                    "Add next-hop address case.";
                  leaf next-hop-address {
                    type inet:ipv6-address;
```

```
                    description
                      "IPv6 address of the next-hop.";
                  }
                }
              }
            }
          }
        }
      }

      /* RPC operations */

      augment "/rt:fib-route/rt:input/rt:destination-address" {
        when "rt:address-family='v6ur:ipv6-unicast'" {
          description
            "This augment is valid only for IPv6 unicast.";
        }
        description
          "This leaf augments the 'rt:destination-address' parameter of
           the 'rt:fib-route' operation.";
        leaf address {
          type inet:ipv6-address;
          description
            "IPv6 destination address.";
        }
      }

      augment "/rt:fib-route/rt:output/rt:route" {
        when "rt:address-family='v6ur:ipv6-unicast'" {
          description
            "This augment is valid only for IPv6 unicast.";
        }
        description
          "This leaf augments the reply to the 'rt:fib-route'
           operation.";
        leaf destination-prefix {
          type inet:ipv6-prefix;
          description
            "IPv6 destination prefix.";
        }
      }

      augment "/rt:fib-route/rt:output/rt:route/rt:next-hop/"
            + "rt:next-hop-options/rt:simple-next-hop" {
        when "../rt:address-family='v6ur:ipv6-unicast'" {
          description
            "This augment is valid only for IPv6 unicast.";
        }
```

```
      description
        "This leaf augments the 'simple-next-hop' case in the reply to
         the 'rt:fib-route' operation.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }
  }

  <CODE ENDS>
```

## 10. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URIs in the IETF XML registry [RFC3688]:

```
--------------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-routing

Registrant Contact: The IESG.

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

--------------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing

Registrant Contact: The IESG.

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

--------------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing

Registrant Contact: The IESG.

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

This document registers the following YANG modules in the YANG Module Names registry [RFC6020]:

```
--------------------------------------------------------------------
name:         ietf-routing
namespace:    urn:ietf:params:xml:ns:yang:ietf-routing
prefix:       rt
reference:    RFC XXXX
--------------------------------------------------------------------

--------------------------------------------------------------------
name:         ietf-ipv4-unicast-routing
namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
prefix:       v4ur
reference:    RFC XXXX
--------------------------------------------------------------------

--------------------------------------------------------------------
name:         ietf-ipv6-unicast-routing
namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
prefix:       v6ur
reference:    RFC XXXX
--------------------------------------------------------------------
```

## 11. Security Considerations

Configuration and state data conforming to the core routing data model (defined in this document) are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

A number of data nodes defined in the YANG modules belonging to the configuration part of the core routing data model are writable/creatable/deletable (i.e., "config true" in YANG terms, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes, such as "edit-config", can have negative effects on the network if the protocol operations are not properly protected.

The vulnerable "config true" subtrees and data nodes are the following:

/routing/routing-instance/interfaces/interface: This list assigns a network layer interface to a routing instance and may also specify interface parameters related to routing.

/routing/routing-instance/routing-protocols/routing-protocol: This list specifies the routing protocols configured on a device.

/routing/ribs/rib: This list specifies the RIBs configured for the device.

Unauthorized access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

## 12. Acknowledgments

The authors wish to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Jeff Haas, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Stephane Litkowski, Thomas Morin, Tom Petch, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler, Yi Yang, Derek Man-Kit Yeung and Jeffrey Zhang for their helpful comments and suggestions.

## 13. References

### 13.1. 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.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

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

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

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, June 2014.

## 13.2. Informative References

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, February 2006.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

# Appendix A. The Complete Data Trees

This appendix presents the complete configuration and state data trees of the core routing data model.

See Section 2.2 for an explanation of the symbols used. Data type of every leaf node is shown near the right end of the corresponding line.

## A.1. Configuration Data

```
+--rw routing
   +--rw routing-instance* [name]
   |  +--rw name                 string
   |  +--rw type?                identityref
   |  +--rw enabled?             boolean
   |  +--rw router-id?           yang:dotted-quad
   |  +--rw description?         string
   |  +--rw default-ribs {multiple-ribs}?
   |  |  +--rw default-rib* [address-family]
   |  |     +--rw address-family    identityref
   |  |     +--rw rib-name          string
   |  +--rw interfaces
   |  |  +--rw interface* [name]
   |  |     +--rw name                              if:interface-ref
   |  |     +--rw v6ur:ipv6-router-advertisements
   |  |        +--rw v6ur:send-advertisements?    boolean
   |  |        +--rw v6ur:max-rtr-adv-interval?   uint16
   |  |        +--rw v6ur:min-rtr-adv-interval?   uint16
   |  |        +--rw v6ur:managed-flag?           boolean
```

```
|  |        +--rw v6ur:other-config-flag?      boolean
|  |        +--rw v6ur:link-mtu?               uint32
|  |        +--rw v6ur:reachable-time?         uint32
|  |        +--rw v6ur:retrans-timer?          uint32
|  |        +--rw v6ur:cur-hop-limit?          uint8
|  |        +--rw v6ur:default-lifetime?       uint16
|  |        +--rw v6ur:prefix-list
|  |           +--rw v6ur:prefix* [prefix-spec]
|  |              +--rw v6ur:prefix-spec           inet:ipv6-prefix
|  |              +--rw (control-adv-prefixes)?
|  |                 +--:(no-advertise)
|  |                 |  +--rw v6ur:no-advertise?         empty
|  |                 +--:(advertise)
|  |                    +--rw v6ur:valid-lifetime?       uint32
|  |                    +--rw v6ur:on-link-flag?         boolean
|  |                    +--rw v6ur:preferred-lifetime?   uint32
|  |                    +--rw v6ur:autonomous-flag?      boolean
|  +--rw routing-protocols
|     +--rw routing-protocol* [type name]
|        +--rw type                identityref
|        +--rw name                string
|        +--rw description?        string
|        +--rw enabled?            boolean
|        +--rw route-preference?   route-preference
|        +--rw connected-ribs
|        |  +--rw connected-rib* [rib-name]
|        |     +--rw rib-name    rib-ref
|        +--rw static-routes
|           +--rw v6ur:ipv6
|           |  +--rw v6ur:route* [destination-prefix]
|           |     +--rw v6ur:destination-prefix    inet:ipv6-prefix
|           |     +--rw v6ur:description?          string
|           |     +--rw v6ur:next-hop
|           |        +--rw (next-hop-options)
|           |           +--:(simple-next-hop)
|           |           |  +--rw v6ur:outgoing-interface?
|           |           +--:(special-next-hop)
|           |           |  +--rw v6ur:special-next-hop?   enumeration
|           |           +--:(next-hop-address)
|           |              +--rw v6ur:next-hop-address?
|           +--rw v4ur:ipv4
|              +--rw v4ur:route* [destination-prefix]
|                 +--rw v4ur:destination-prefix    inet:ipv4-prefix
|                 +--rw v4ur:description?          string
|                 +--rw v4ur:next-hop
|                    +--rw (next-hop-options)
|                       +--:(simple-next-hop)
|                       |  +--rw v4ur:outgoing-interface?
```

```
   |                       +--:(special-next-hop)
   |                       |  +--rw v4ur:special-next-hop?   enumeration
   |                       +--:(next-hop-address)
   |                          +--rw v4ur:next-hop-address?
   +--rw ribs
      +--rw rib* [name]
         +--rw name              string
         +--rw address-family    identityref
         +--rw description?      string
         +--rw recipient-ribs {multiple-ribs}?
            +--rw recipient-rib* [rib-name]
               +--rw rib-name    rib-ref
```

A.2.  State Data

```
  +--ro routing-state
     +--ro routing-instance* [name]
     |  +--ro name                 string
     |  +--ro type?                identityref
     |  +--ro default-ribs
     |  |  +--ro default-rib* [address-family]
     |  |     +--ro address-family    identityref
     |  |     +--ro rib-name          rib-state-ref
     |  +--ro interfaces
     |  |  +--ro interface* [name]
     |  |     +--ro name                          if:interface-state-ref
     |  |     +--ro v6ur:ipv6-router-advertisements
     |  |        +--ro v6ur:send-advertisements?    boolean
     |  |        +--ro v6ur:max-rtr-adv-interval?   uint16
     |  |        +--ro v6ur:min-rtr-adv-interval?   uint16
     |  |        +--ro v6ur:managed-flag?           boolean
     |  |        +--ro v6ur:other-config-flag?      boolean
     |  |        +--ro v6ur:link-mtu?               uint32
     |  |        +--ro v6ur:reachable-time?         uint32
     |  |        +--ro v6ur:retrans-timer?          uint32
     |  |        +--ro v6ur:cur-hop-limit?          uint8
     |  |        +--ro v6ur:default-lifetime?       uint16
     |  |        +--ro v6ur:prefix-list
     |  |           +--ro v6ur:prefix* [prefix-spec]
     |  |              +--ro v6ur:prefix-spec           inet:ipv6-prefix
     |  |              +--ro v6ur:valid-lifetime?       uint32
     |  |              +--ro v6ur:on-link-flag?         boolean
     |  |              +--ro v6ur:preferred-lifetime?   uint32
     |  |              +--ro v6ur:autonomous-flag?      boolean
     |  +--ro routing-protocols
     |     +--ro routing-protocol* [type name]
     |        +--ro type                identityref
     |        +--ro name                string
```

```
     |        +--ro route-preference    route-preference
     |        +--ro connected-ribs
     |           +--ro connected-rib* [rib-name]
     |              +--ro rib-name    rib-state-ref
     +--ro ribs
        +--ro rib* [name]
           +--ro name              string
           +--ro address-family    identityref
           +--ro routes
           |  +--ro route*
           |     +--ro route-preference?          route-preference
           |     +--ro next-hop
           |     |  +--ro (next-hop-options)
           |     |     +--:(simple-next-hop)
           |     |     |  +--ro outgoing-interface?
           |     |     |  +--ro v6ur:next-hop-address? inet:ipv6-address
           |     |     |  +--ro v4ur:next-hop-address? inet:ipv4-address
           |     |     +--:(special-next-hop)
           |     |        +--ro special-next-hop?        enumeration
           |     +--ro source-protocol            identityref
           |     +--ro active?                    empty
           |     +--ro last-updated?              yang:date-and-time
           |     +--ro v6ur:destination-prefix?   inet:ipv6-prefix
           |     +--ro v4ur:destination-prefix?   inet:ipv4-prefix
           +--ro recipient-ribs
              +--ro recipient-rib* [rib-name]
                 +--ro rib-name    rib-state-ref
```

## Appendix B. Minimum Implementation

Some parts and options of the core routing model, such as user-defined routing tables, are intended only for advanced routers. This appendix gives basic non-normative guidelines for implementing a bare minimum of available functions. Such an implementation may be used for hosts or very simple routers.

A minimum implementation will provide a single system-controlled routing instance, and will not allow clients to create any user-controlled instances.

Typically, the feature "multiple-ribs" will not be supported. This means that a single system-controlled RIB is available for each supported address family - IPv4, IPv6 or both. These RIBs must be the default RIBs, so references to them will also appear as system-controlled entries of the "default-rib" list in state data. No user-controlled RIBs are allowed.

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configured instance(s) of the "static" pseudo-protocol. Even with a single RIB per address family, it may be occasionally useful to be able to configure multiple "static" instances. For example, a client may want to configure alternative sets of static routes and activate or deactivate them by means of connnecting the default RIB to the corresponding "static" instance.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" routing protocol instances.

## Appendix C. Example: Adding a New Routing Protocol

This appendix demonstrates how the core routing data model can be extended to support a new routing protocol. The YANG module "example-rip" shown below is intended only as an illustration rather than a real definition of a data model for the RIP routing protocol. For the sake of brevity, this module does not obey all the guidelines specified in [RFC6087]. See also Section 5.4.2.

```
module example-rip {

  namespace "http://example.com/rip";

  prefix "rip";

  import ietf-routing {
    prefix "rt";
  }

  identity rip {
    base rt:routing-protocol;
    description
      "Identity for the RIP routing protocol.";
  }

  typedef rip-metric {
    type uint8 {
      range "0..16";
    }
  }

  grouping route-content {
    description
      "This grouping defines RIP-specific route attributes.";
    leaf metric {
```

```
        type rip-metric;
      }
      leaf tag {
        type uint16;
        default "0";
        description
          "This leaf may be used to carry additional info, e.g. AS
           number.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
      when "rt:source-protocol = 'rip:rip'" {
        description
          "This augment is only valid for a routes whose source
           protocol is RIP.";
      }
      description
        "RIP-specific route attributes.";
      uses route-content;
    }

    augment "/rt:active-route/rt:output/rt:route" {
      description
        "RIP-specific route attributes in the output of 'active-route'
         RPC.";
      uses route-content;
    }

    augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
          + "rt:routing-protocol" {
      when "rt:type = 'rip:rip'" {
        description
          "This augment is only valid for a routing protocol instance
           of type 'rip'.";
      }
      container rip {
        description
          "RIP instance configuration.";
        container interfaces {
          description
            "Per-interface RIP configuration.";
          list interface {
            key "name";
            description
              "RIP is enabled on interfaces that have an entry in this
               list, unless 'enabled' is set to 'false' for that
               entry.";
```

```
              leaf name {
                type leafref {
                  path "../../../../../../rt:interfaces/rt:interface/"
                     + "rt:name";
                }
              }
              leaf enabled {
                type boolean;
                default "true";
              }
              leaf metric {
                type rip-metric;
                default "1";
              }
            }
          }
          leaf update-interval {
            type uint8 {
              range "10..60";
            }
            units "seconds";
            default "30";
            description
              "Time interval between periodic updates.";
          }
        }
      }
    }
```

## Appendix D. Example: NETCONF <get> Reply

This section contains a sample reply to the NETCONF <get> message, which could be sent by a server supporting (i.e., advertising them in the NETCONF <hello> message) the following YANG modules:

- ietf-interfaces [RFC7223],
- ietf-ip [RFC7277],
- ietf-routing (Section 7),
- ietf-ipv4-unicast-routing (Section 8),
- ietf-ipv6-unicast-routing (Section 9).

We assume a simple network set-up as shown in Figure 4: router "A" uses static default routes with the "ISP" router as the next-hop.

IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

```
                 +-----------------+
                 |                 |
                 |    Router ISP   |
                 |                 |
                 +--------+--------+
                          |2001:db8:0:1::2
                          |192.0.2.2
                          |
                          |
                          |2001:db8:0:1::1
                     eth0|192.0.2.1
                 +--------+--------+
                 |                 |
                 |     Router A    |
                 |                 |
                 +--------+--------+
                     eth1|198.51.100.1
                          |2001:db8:0:2::1
                          |
```

Figure 4: Example network configuration

A reply to the NETCONF <get> message sent by router "A" would then be as follows:

```
<?xml version="1.0"?>
<rpc-reply
    message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:v4ur="urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing"
    xmlns:v6ur="urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing"
    xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
    xmlns:ip="urn:ietf:params:xml:ns:yang:ietf-ip"
    xmlns:rt="urn:ietf:params:xml:ns:yang:ietf-routing">
 <data>
  <if:interfaces>
   <if:interface>
    <if:name>eth0</if:name>
    <if:type>ianaift:ethernetCsmacd</if:type>
    <if:description>
     Uplink to ISP.
    </if:description>
    <ip:ipv4>
     <ip:address>
```

```
        <ip:ip>192.0.2.1</ip:ip>
        <ip:prefix-length>24</ip:prefix-length>
       </ip:address>
       <ip:forwarding>true</ip:forwarding>
      </ip:ipv4>
      <ip:ipv6>
       <ip:address>
        <ip:ip>2001:0db8:0:1::1</ip:ip>
        <ip:prefix-length>64</ip:prefix-length>
       </ip:address>
       <ip:forwarding>true</ip:forwarding>
       <ip:autoconf>
        <ip:create-global-addresses>false</ip:create-global-addresses>
       </ip:autoconf>
      </ip:ipv6>
     </if:interface>
     <if:interface>
      <if:name>eth1</if:name>
      <if:type>ianaift:ethernetCsmacd</if:type>
      <if:description>
       Interface to the internal network.
      </if:description>
      <ip:ipv4>
       <ip:address>
        <ip:ip>198.51.100.1</ip:ip>
        <ip:prefix-length>24</ip:prefix-length>
       </ip:address>
       <ip:forwarding>true</ip:forwarding>
      </ip:ipv4>
      <ip:ipv6>
       <ip:address>
        <ip:ip>2001:0db8:0:2::1</ip:ip>
        <ip:prefix-length>64</ip:prefix-length>
       </ip:address>
       <ip:forwarding>true</ip:forwarding>
       <ip:autoconf>
        <ip:create-global-addresses>false</ip:create-global-addresses>
       </ip:autoconf>
      </ip:ipv6>
     </if:interface>
    </if:interfaces>
    <if:interfaces-state>
     <if:interface>
      <if:name>eth0</if:name>
      <if:type>ianaift:ethernetCsmacd</if:type>
      <if:phys-address>00:0C:42:E5:B1:E9</if:phys-address>
      <if:oper-status>up</if:oper-status>
      <if:statistics>
```

```
        <if:discontinuity-time>
         2014-10-24T17:11:27+00:58
        </if:discontinuity-time>
       </if:statistics>
       <ip:ipv4>
        <ip:forwarding>true</ip:forwarding>
        <ip:mtu>1500</ip:mtu>
        <ip:address>
         <ip:ip>192.0.2.1</ip:ip>
         <ip:prefix-length>24</ip:prefix-length>
        </ip:address>
       </ip:ipv4>
       <ip:ipv6>
        <ip:forwarding>true</ip:forwarding>
        <ip:mtu>1500</ip:mtu>
        <ip:address>
         <ip:ip>2001:0db8:0:1::1</ip:ip>
         <ip:prefix-length>64</ip:prefix-length>
        </ip:address>
       </ip:ipv6>
      </if:interface>
      <if:interface>
       <if:name>eth1</if:name>
       <if:type>ianaift:ethernetCsmacd</if:type>
       <if:oper-status>up</if:oper-status>
       <if:phys-address>00:0C:42:E5:B1:EA</if:phys-address>
       <if:statistics>
        <if:discontinuity-time>
         2014-10-24T17:11:27+00:59
        </if:discontinuity-time>
       </if:statistics>
       <ip:ipv4>
        <ip:forwarding>true</ip:forwarding>
        <ip:mtu>1500</ip:mtu>
        <ip:address>
         <ip:ip>198.51.100.1</ip:ip>
         <ip:prefix-length>24</ip:prefix-length>
        </ip:address>
       </ip:ipv4>
       <ip:ipv6>
        <ip:forwarding>true</ip:forwarding>
        <ip:mtu>1500</ip:mtu>
        <ip:address>
         <ip:ip>2001:0db8:0:2::1</ip:ip>
         <ip:prefix-length>64</ip:prefix-length>
        </ip:address>
       </ip:ipv6>
      </if:interface>
```

```
    </if:interfaces-state>
    <rt:routing>
     <rt:routing-instance>
      <rt:name>rtr0</rt:name>
      <rt:description>Router A</rt:description>
      <rt:interfaces>
       <rt:interface>
        <rt:name>eth1</rt:name>
        <v6ur:ipv6-router-advertisements>
         <v6ur:send-advertisements>true</v6ur:send-advertisements>
         <v6ur:prefix-list>
          <v6ur:prefix>
           <v6ur:prefix-spec>2001:db8:0:2::/64</v6ur:prefix-spec>
          </v6ur:prefix>
         </v6ur:prefix-list>
        </v6ur:ipv6-router-advertisements>
       </rt:interface>
      </rt:interfaces>
      <rt:routing-protocols>
       <rt:routing-protocol>
        <rt:type>rt:static</rt:type>
        <rt:name>st0</rt:name>
        <rt:description>
         Static routing is used for the internal network.
        </rt:description>
        <rt:static-routes>
         <v4ur:ipv4>
          <v4ur:route>
           <v4ur:destination-prefix>0.0.0.0/0</v4ur:destination-prefix>
           <v4ur:next-hop>
        <v4ur:next-hop-address>192.0.2.2</v4ur:next-hop-address>
       </v4ur:next-hop>
          </v4ur:route>
         </v4ur:ipv4>
         <v6ur:ipv6>
          <v6ur:route>
           <v6ur:destination-prefix>::/0</v6ur:destination-prefix>
           <v6ur:next-hop>
        <v6ur:next-hop-address>2001:db8:0:1::2</v6ur:next-hop-address>
       </v6ur:next-hop>
          </v6ur:route>
         </v6ur:ipv6>
        </rt:static-routes>
       </rt:routing-protocol>
      </rt:routing-protocols>
     </rt:routing-instance>
    </rt:routing>
    <rt:routing-state>
```

```
<rt:routing-instance>
 <rt:name>rtr0</rt:name>
 <rt:default-ribs>
  <rt:default-rib>
   <rt:address-family>v4ur:ipv4-unicast</rt:address-family>
   <rt:rib-name>ipv4-master</rt:rib-name>
  </rt:default-rib>
  <rt:default-rib>
   <rt:address-family>v6ur:ipv6-unicast</rt:address-family>
   <rt:rib-name>ipv6-master</rt:rib-name>
  </rt:default-rib>
 </rt:default-ribs>
 <rt:interfaces>
  <rt:interface>
   <rt:name>eth0</rt:name>
  </rt:interface>
  <rt:interface>
   <rt:name>eth1</rt:name>
   <v6ur:ipv6-router-advertisements>
    <v6ur:send-advertisements>true</v6ur:send-advertisements>
    <v6ur:prefix-list>
     <v6ur:prefix>
      <v6ur:prefix-spec>2001:db8:0:2::/64</v6ur:prefix-spec>
     </v6ur:prefix>
    </v6ur:prefix-list>
   </v6ur:ipv6-router-advertisements>
  </rt:interface>
 </rt:interfaces>
 <rt:routing-protocols>
  <rt:routing-protocol>
   <rt:type>rt:static</rt:type>
   <rt:name>st0</rt:name>
   <rt:route-preference>5</rt:route-preference>
  </rt:routing-protocol>
 </rt:routing-protocols>
</rt:routing-instance>
<rt:ribs>
 <rt:rib>
  <rt:name>ipv4-master</rt:name>
  <rt:address-family>v4ur:ipv4-unicast</rt:address-family>
  <rt:routes>
   <rt:route>
    <v4ur:destination-prefix>192.0.2.1/24</v4ur:destination-prefix>
    <rt:next-hop>
 <rt:outgoing-interface>eth0</rt:outgoing-interface>
    </rt:next-hop>
    <rt:route-preference>0</rt:route-preference>
    <rt:source-protocol>rt:direct</rt:source-protocol>
```

```
         <rt:last-updated>2014-10-24T17:11:27+01:00</rt:last-updated>
        </rt:route>
        <rt:route>
      <v4ur:destination-prefix>198.51.100.0/24</v4ur:destination-prefix>
         <rt:next-hop>
      <rt:outgoing-interface>eth1</rt:outgoing-interface>
         </rt:next-hop>
         <rt:source-protocol>rt:direct</rt:source-protocol>
         <rt:route-preference>0</rt:route-preference>
         <rt:last-updated>2014-10-24T17:11:27+01:00</rt:last-updated>
        </rt:route>
        <rt:route>
         <v4ur:destination-prefix>0.0.0.0/0</v4ur:destination-prefix>
         <rt:source-protocol>rt:static</rt:source-protocol>
         <rt:route-preference>5</rt:route-preference>
         <rt:next-hop>
      <v4ur:next-hop-address>192.0.2.2</v4ur:next-hop-address>
         </rt:next-hop>
         <rt:last-updated>2014-10-24T18:02:45+01:00</rt:last-updated>
        </rt:route>
       </rt:routes>
      </rt:rib>
      <rt:rib>
       <rt:name>ipv6-master</rt:name>
       <rt:address-family>v6ur:ipv6-unicast</rt:address-family>
       <rt:routes>
        <rt:route>
         <v6ur:destination-prefix>
      2001:db8:0:1::/64
         </v6ur:destination-prefix>
         <rt:next-hop>
      <rt:outgoing-interface>eth0</rt:outgoing-interface>
         </rt:next-hop>
         <rt:source-protocol>rt:direct</rt:source-protocol>
         <rt:route-preference>0</rt:route-preference>
         <rt:last-updated>2014-10-24T17:11:27+01:00</rt:last-updated>
        </rt:route>
        <rt:route>
         <v6ur:destination-prefix>
      2001:db8:0:2::/64
         </v6ur:destination-prefix>
         <rt:next-hop>
      <rt:outgoing-interface>eth1</rt:outgoing-interface>
         </rt:next-hop>
         <rt:source-protocol>rt:direct</rt:source-protocol>
         <rt:route-preference>0</rt:route-preference>
         <rt:last-updated>2014-10-24T17:11:27+01:00</rt:last-updated>
        </rt:route>
```

```
        <rt:route>
         <v6ur:destination-prefix>::/0</v6ur:destination-prefix>
         <rt:next-hop>
      <v6ur:next-hop-address>2001:db8:0:1::2</v6ur:next-hop-address>
         </rt:next-hop>
         <rt:source-protocol>rt:static</rt:source-protocol>
         <rt:route-preference>5</rt:route-preference>
         <rt:last-updated>2014-10-24T18:02:45+01:00</rt:last-updated>
        </rt:route>
       </rt:routes>
      </rt:rib>
     </rt:ribs>
    </rt:routing-state>
   </data>
  </rpc-reply>
```

## Appendix E. Change Log

RFC Editor: Remove this section upon publication as an RFC.

### E.1. Changes Between Versions -16 and -17

- Added Acee as a co-author.
- Removed all traces of route filters.
- Removed numeric IDs of list entries in state data.
- Removed all next-hop cases except "simple-next-hop" and "special-next-hop".
- Removed feature "multipath-routes".
- Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing form state data of an interface entry to the routing instance(s) to which the interface is assigned.

### E.2. Changes Between Versions -15 and -16

- Added ’type’ as the second key component of ’routing-protocol’, both in configuration and state data.
- The restriction of no more than one connected RIB per address family was removed.
- Removed the ’id’ key of routes in RIBs. This list has no keys anymore.

- o Remove the 'id' key from static routes and make 'destination-prefix' the only key.

- o Added 'route-preference' as a new attribute of routes in RIB.

- o Added 'active' as a new attribute of routes in RIBs.

- o Renamed RPC operation 'active-route' to 'fib-route'.

- o Added 'route-preference' as a new parameter of routing protocol instances, both in configuration and state data.

- o Renamed identity 'rt:standard-routing-instance' to 'rt:default-routing-instance'.

- o Added next-hop lists to state data.

- o Added two cases for specifying next-hops indirectly - via a new RIB or a recursive list of next-hops.

- o Reorganized next-hop in static routes.

- o Removed all 'if-feature' statements from state data.

## E.3. Changes Between Versions -14 and -15

- o Removed all defaults from state data.

- o Removed default from 'cur-hop-limit' in config.

## E.4. Changes Between Versions -13 and -14

- o Removed dependency of 'connected-ribs' on the 'multiple-ribs' feature.

- o Removed default value of 'cur-hop-limit' in state data.

- o Moved parts of descriptions and all references on IPv6 RA parameters from state data to configuration.

- o Added reference to RFC 6536 in the Security section.

## E.5. Changes Between Versions -12 and -13

- o Wrote appendix about minimum implementation.

- o Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.

- o Extra container for the next-hop list.
- o Names rather than numeric ids are used for referring to list entries in state data.
- o Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- o Descriptions of "name" keys in state data lists are required to be persistent.
- o
- o Removed "if-feature multiple-ribs;" from connected-ribs.
- o "rib-name" instead of "name" is used as the name of leafref nodes.
- o "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

## E.6. Changes Between Versions -11 and -12

- o Removed feature "advanced-router" and introduced two features instead: "multiple-ribs" and "multipath-routes".
- o Unified the keys of config and state versions of "routing-instance" and "rib" lists.
- o Numerical identifiers of state list entries are not keys anymore, but they are constrained using the "unique" statement.
- o Updated acknowledgements.

## E.7. Changes Between Versions -10 and -11

- o Migrated address families from IANA enumerations to identities.
- o Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- o Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- o Described the relationship between system-controlled and user-controlled list entries.
- o Feature "user-defined-routing-tables" changed into "advanced-router".

- o Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).

- o Added nexthop-list with entries having priorities (backup) and weights (load balancing).

- o Updated bibliography references.

## E.8. Changes Between Versions -09 and -10

- o Added subtree for state data ("/routing-state").

- o Terms "system-controlled entry" and "user-controlled entry" defined and used.

- o New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.

- o Added grouping "router-id".

- o In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".

- o Renamed "main-routing-table" to "default-routing-table".

## E.9. Changes Between Versions -08 and -09

- o Fixed "must" expresion for "connected-routing-table".

- o Simplified "must" expression for "main-routing-table".

- o Moved per-interface configuration of a new routing protocol under ’routing-protocol’. This also affects the ’example-rip’ module.

## E.10. Changes Between Versions -07 and -08

- o Changed reference from RFC6021 to RFC6021bis.

## E.11. Changes Between Versions -06 and -07

- o The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.

- o The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.

- o The default behavior for IPv6 RA prefix advertisements was clarified.

- o Changed type of "rt:router-id" to "ip:dotted-quad".

- o Type of "rt:router-id" changed to "yang:dotted-quad".

- o Fixed missing prefixes in XPath expressions.

## E.12. Changes Between Versions -05 and -06

- o Document title changed: "Configuration" was replaced by "Management".

- o New typedefs "routing-table-ref" and "route-filter-ref".

- o Double slashes "//" were removed from XPath expressions and replaced with the single "/".

- o Removed uniqueness requirement for "router-id".

- o Complete data tree is now in Appendix A.

- o Changed type of "source-protocol" from "leafref" to "string".

- o Clarified the relationship between routing protocol instances and connected routing tables.

- o Added a must constraint saying that a routing table connected to the direct pseudo-protocol must not be a main routing table.

## E.13. Changes Between Versions -04 and -05

- o Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".

- o "must" statement for "static-routes" changed to "when".

- o Added "main-routing-tables" containing references to main routing tables for each address family.

- o Removed the defaults for "address-family" and "safi" and made them mandatory.

- o Removed the default for route-filter/type and made this leaf mandatory.

- o If there is no active route for a given destination, the "active-route" RPC returns no output.

- o Added "enabled" switch under "routing-protocol".

- o Added "router-type" identity and "type" leaf under "router".

- o Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".

- o The "direct" pseudo-protocol is always connected to main routing tables.

- o Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".

- o Added "must" constraint saying that a routing table must not be its own recipient.

## E.14. Changes Between Versions -03 and -04

- o Changed "error-tag" for both RPC operations from "missing element" to "data-missing".

- o Removed the decrementing behavior for advertised IPv6 prefix parameters "valid-lifetime" and "preferred-lifetime".

- o Changed the key of the static route lists from "seqno" to "id" because the routes needn’t be sorted.

- o Added ’must’ constraint saying that "preferred-lifetime" must not be greater than "valid-lifetime".

## E.15. Changes Between Versions -02 and -03

- o Module "iana-afn-safi" moved to I-D "iana-if-type".

- o Removed forwarding table.

- o RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).

- o New RPC "route-count".

- o For both RPCs, specification of negative responses was added.

- o Relaxed separation of router instances.

- o Assignment of interfaces to router instances needn’t be disjoint.
- o Route filters are now global.
- o Added "allow-all-route-filter" for symmetry.
- o Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- o Added "router-id" leaf.
- o Specified the names for IPv4/IPv6 unicast main routing tables.
- o Route parameter "last-modified" changed to "age".
- o Added container "recipient-routing-tables".

## E.16. Changes Between Versions -01 and -02

- o Added module "ietf-ipv6-unicast-routing".
- o The example in Appendix D now uses IP addresses from blocks reserved for documentation.
- o Direct routes appear by default in the forwarding table.
- o Network layer interfaces must be assigned to a router instance. Additional interface configuration may be present.
- o The "when" statement is only used with "augment", "must" is used elsewhere.
- o Additional "must" statements were added.
- o The "route-content" grouping for IPv4 and IPv6 unicast now includes the material from the "ietf-routing" version via "uses rt:route-content".
- o Explanation of symbols in the tree representation of data model hierarchy.

## E.17. Changes Between Versions -00 and -01

- o AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- o Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.

- o Names of some data nodes were changed, in particular "routing-process" is now "router".
- o The restriction of a single AFN/SAFI per router was lifted.
- o RPC operation "delete-route" was removed.
- o Illegal XPath references from "get-route" to the datastore were fixed.
- o Section "Security Considerations" was written.

## Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Lindem
Cisco Systems

Email: acee@cisco.com

NETMOD WG
Internet-Draft
Intended status: Informational
Expires: Sep 09, 2015

Clyde Wildes
Cisco Systems
Agrahara Kiran Koushik
Brocade Communication Systems

Mar 09, 2015

# SYSLOG YANG model
# draft-ietf-netmod-syslog-model-03

## Abstract

This document describes a data model for Syslog protocol which is used to convey event notification messages.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

Operating systems, processes and applications generate messages indicating their own status or the occurrence of events. These messages are useful for managing and/or debugging the network and its services. The BSD Syslog protocol is a widely adopted protocol that is used for transmission and processing of the messages.

Since each process, application and operating system was written somewhat independently, there is little uniformity to the content of Syslog messages. For this reason, no assumption is made upon the formatting or contents of the messages. The protocol is simply designed to transport these event messages. No acknowledgement of the receipt is made.

Essentially, a Syslog process receives messages (from the kernel, processes, applications or other Syslog processes) and processes those. The processing involves logging to a local file, displaying on console, user terminal, and/or relaying to syslog processes on other machines. The processing is determined by the "facility" that originated the message and the "severity" assigned to the message by the facility.

We are using definitions of Syslog protocol from [RFC3164] in this draft.

### 1.1. Definitions and Acronyms

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

UDP: User Datagram Protocol

VRF: Virtual Routing and Forwarding

## 2. Problem Statement

This document defines a YANG [RFC6020] configuration data model that may be used to monitor and control one or more syslog processes running on a system. YANG models can be used with network management agents such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

This module makes use of the YANG "feature" construct which allows implementations to support only those Syslog features that lie within their capabilities.

## 3. Design of the SYSLOG Model

The syslog model was designed by comparing various syslog features implemented by various vendors’ in different implementations.

This draft addresses the common leafs between all vendors and creates a common model, which can be augmented with proprietary features, if necessary. The base model is designed to be very simple for maximum flexibility.

Syslog consists of message producers, a group level suppression filter, and message distributors. The following digram shows syslog messages flowing from a message producer, through the group level suppression filter, and if passed by the group filter to message distributors where further suppression filtering can take place.

```
   Message Producers
    +-------------+  +-------------+  +-------------+  +-------------+
    |  Various    |  |     OS      |  |             |  |   Remote    |
    | Components  |  |   Kernel    |  | Line Cards  |  |   Servers   |
    +-------------+  +-------------+  +-------------+  +-------------+

    +-------------+  +-------------+  +-------------+  +-------------+
    |    SNMP     |  |  Interface  |  |   Standby   |  |   Syslog    |
    |   Events    |  |   Events    |  |  Supervisor |  |   Itself    |
    +-------------+  +-------------+  +-------------+  +-------------+

    |                                                                 |
    +-----------------------------------------------------------------+
                 |
                 |
                 v

   Group Level Suppression
    +-------------------------------+
    | Filter by message facility    |
    | and message severity          |
    +-------------------------------+
                 |
                 |
                 |
         +------------+------------+------------+-----------+
         |            |            |            |           |
         v            v            v            v           v
   Message Distributors
    +----------+ +----------+ +----------+  +----------+  +----------+
    |          | | Log      | | Log      |  |    User  |  |   Remote |
    |  Console | | Buffer   | | File(s)  |  | Terminals|  |   Servers|
    +----------+ +----------+ +----------+  +----------+  +----------+
```

The leaves in the base syslog model correspond to the group level suppression filter and each message distributor:

- console
- log buffer
- log file(s)
- user terminals
- remote server(s).

Optional features are used to specified fields that are not present in all vendor configurations.

3.1.  SYSLOG Module

```
module: ietf-syslog
   +--rw syslog
      +--rw global-logging-action {global-logging-action}?
      |  +--rw (logging-level-scope)?
      |  |  +--:(logging-facility-all)
      |  |  |  +--rw severity?                          syslogtypes:severity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                              empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw severity?   syslogtypes:severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw select-message-severity?   enumeration
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |     +--rw pattern-match?   string
      +--rw console-logging-action
      |  +--rw (logging-level-scope)?
      |  |  +--:(logging-facility-all)
      |  |  |  +--rw severity?                          syslogtypes:severity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                              empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw severity?   syslogtypes:severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw select-message-severity?   enumeration
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |     +--rw pattern-match?   string
      +--rw buffered-logging-action
      |  +--rw (logging-level-scope)?
      |  |  +--:(logging-facility-all)
      |  |  |  +--rw severity?                          syslogtypes:severity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                              empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw severity?   syslogtypes:severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw select-message-severity?   enumeration
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |  |  +--rw pattern-match?   string
      |  +--rw buffer-size?                       uint64
      +--rw file-logging-action
      |  +--rw logging-files* [file-name]
      |     +--rw file-name                          inet:uri
      |     +--rw (logging-level-scope)?
      |     |  +--:(logging-facility-all)
      |     |  |  +--rw severity?                          syslogtypes:severit
y
      |     |  +--:(logging-facility-none)
      |     |  |  +--rw none?                              empty
      |     |  +--:(logging-facility)
      |     |     +--rw logging-facilities* [facility]
      |     |        +--rw facility    identityref
      |     |        +--rw severity?   syslogtypes:severity
      |     +--rw logging-advanced-level-processing {selector-advanced-level-pro
cessing-config}?
```

```
      |     |  +--rw select-message-severity?   enumeration
      |     +--rw logging-match-processing {selector-match-processing-config}?
      |     |  +--rw pattern-match?   string
      |     +--rw file-logging-structured-data?        boolean {file-logging-str
uctured-data}?
      |     +--rw file-logging-archive {file-logging-archive-config}?
      |        +--rw file-number?       uint32
      |        +--rw file-size?         uint64
      |        +--rw file-permission?   enumeration
      +--rw remote-logging-action
      |  +--rw remote-logging-destination* [destination]
      |     +--rw destination                          inet:host
      |     +--rw (logging-level-scope)?
      |     |  +--:(logging-facility-all)
      |     |  |  +--rw severity?                             syslogtypes:severit
y
      |     |  +--:(logging-facility-none)
      |     |  |  +--rw none?                                 empty
      |     |  +--:(logging-facility)
      |     |     +--rw logging-facilities* [facility]
      |     |        +--rw facility    identityref
      |     |        +--rw severity?   syslogtypes:severity
      |     +--rw logging-advanced-level-processing {selector-advanced-level-pro
cessing-config}?
      |     |  +--rw select-message-severity?   enumeration
      |     +--rw logging-match-processing {selector-match-processing-config}?
      |     |  +--rw pattern-match?   string
      |     +--rw remote-logging-structured-data?      boolean {remote-logging-s
tructured-data}?
      |     +--rw destination-port?                    inet:port-number
      |     +--rw destination-facility?                identityref
      |     +--rw source-interface?                    if:interface-ref
      |     +--rw vrf-name?                            string {remote-logging-us
e-vrf}?
      |     +--rw syslog-sign! {signed-messages-config}?
      |        +--rw cert-initial-repeat    uint16
      |        +--rw cert-resend-delay      uint16
      |        +--rw cert-resend-count      uint16
      |        +--rw sig-max-delay          uint16
      |        +--rw sig-number-resends     uint16
      |        +--rw sig-resend-delay       uint16
      |        +--rw sig-resend-count       uint16
      +--rw terminal-logging-action
         +--rw (user-scope)?
            +--:(all-users)
            |  +--rw all-users
            |     +--rw (logging-level-scope)?
            |     |  +--:(logging-facility-all)
            |     |  |  +--rw severity?                             syslogtypes:s
everity
            |     |  +--:(logging-facility-none)
            |     |  |  +--rw none?                                 empty
            |     |  +--:(logging-facility)
            |     |     +--rw logging-facilities* [facility]
            |     |        +--rw facility    identityref
            |     |        +--rw severity?   syslogtypes:severity
            |     +--rw logging-advanced-level-processing {selector-advanced-lev
el-processing-config}?
            |     |  +--rw select-message-severity?   enumeration
            |     +--rw logging-match-processing {selector-match-processing-conf
ig}?
            |        +--rw pattern-match?   string
            +--:(per-user) {terminal-facility-user-logging-config}?
               +--rw user-name* [uname]
                  +--rw uname                                   string
                  +--rw (logging-level-scope)?
                  |  +--:(logging-facility-all)
```

```
                   |  |  +--rw severity?                            syslogtypes:s
everity
                   |  +--:(logging-facility-none)
                   |  |  +--rw none?                                empty
                   |  +--:(logging-facility)
                   |     +--rw logging-facilities* [facility]
                   |        +--rw facility    identityref
                   |        +--rw severity?   syslogtypes:severity
                   +--rw logging-advanced-level-processing {selector-advanced-lev
el-processing-config}?
                   |  +--rw select-message-severity?   enumeration
                   +--rw logging-match-processing {selector-match-processing-conf
ig}?
                      +--rw pattern-match?   string
```

## 4. SYSLOG YANG Models

### 4.1. SYSLOG-TYPES module

```
<CODE BEGINS> file "ietf-syslog-types.yang"

module ietf-syslog-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog-types";
  prefix syslogtypes;

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

     WG Chair: Tom Nadeau
               <mailto:tnadeau@brocade.com>

     Editor:   Clyde Wildes
               <mailto:cwildes@cisco.com>

     Editor:   Agrahara Kiran Koushik
               <mailto:kkoushik@brocade.com>";
  description
    "This module contains a collection of YANG type definitions for
     SYSLOG.";

  revision 2015-03-05 {
     description
      "Initial Revision";
    reference
      "This model references RFC 5424 - The Syslog Protocol.";


  }

  typedef severity {
    type enumeration {
      enum "emergency" {
        value 0;
        description
          "Emergency Level Msg";
      }
      enum "alert" {
        value 1;
        description
          "Alert Level Msg";
      }
```

```
      enum "critical" {
        value 2;
        description
          "Critical Level Msg";
      }
      enum "error" {
        value 3;
        description
          "Error Level Msg";
      }
      enum "warning" {
        value 4;
        description
          "Warning Level Msg";
      }
      enum "notice" {
        value 5;
        description
          "Notification Level Msg";
      }
      enum "info" {
        value 6;
        description
          "Informational Level Msg";
      }
      enum "debug" {
        value 7;
        description
          "Debugging Level Msg";
      }
    }
    description
      "The definitions for Syslog message severity.";
  }

  identity syslog-facility {
    description
      "The base identity to represent syslog facilities";
  }

  identity kern {
    base syslog-facility;
    description
      "The facility for kernel messages as defined in RFC 5424.";
  }
```

```
  identity user {
    base syslog-facility;
    description
      "The facility for user-level messages as defined in RFC 5424.";
   }

   identity mail {
     base syslog-facility;
    description
      "The facility for the mail system as defined in RFC 5424.";
   }

   identity daemon {
     base syslog-facility;
    description
      "The facility for the system daemons as defined in RFC 5424.";
   }

   identity auth {
     base syslog-facility;
    description
      "The facility for security/authorization messages as defined
       in RFC 5424.";
   }

   identity syslog {
     base syslog-facility;
    description
      "The facility for messages generated internally by syslogd
       facility as defined in RFC 5424.";
   }

   identity lpr {
     base syslog-facility;
    description
      "The facility for the line printer subsystem as defined in
       RFC 5424.";
   }

  identity news {
    base syslog-facility;
    description
      "The facility for the network news subsystem as defined in
       RFC 5424.";
   }

   identity uucp {
     base syslog-facility;
    description
      "The facility for the UUCP subsystem as defined in RFC 5424.";
   }

   identity cron {
     base syslog-facility;
    description
      "The facility for the clock daemon as defined in RFC 5424.";
   }

   identity authpriv {
     base syslog-facility;
    description
      "The facility for privileged security/authorization messages
       as defined in RFC 5424.";
   }
```

```
   identity ftp {
     base syslog-facility;
    description
      "The facility for the FTP daemon as defined in RFC 5424.";
   }

   identity ntp {
     base syslog-facility;
    description
      "The facility for the NTP subsystem as defined in RFC 5424.";
  }

  identity audit {
    base syslog-facility;
    description
      "The facility for log audit messages as defined in RFC 5424.";
   }
```

```
  identity console {
    base syslog-facility;
    description
      "The facility for log alert messages as defined in RFC 5424.";
   }

   identity cron2 {
     base syslog-facility;
    description
      "The facility for the second clock daemon as defined in
       RFC 5424.";
   }

   identity local0 {
     base syslog-facility;
    description
      "The facility for local use 0 messages as defined in
       RFC 5424.";
   }

   identity local1 {
     base syslog-facility;
    description
      "The facility for local use 1 messages as defined in
       RFC 5424.";
   }

   identity local2 {
     base syslog-facility;
    description
      "The facility for local use 2 messages as defined in
       RFC 5424.";
  }

  identity local3 {
    base syslog-facility;
    description
      "The facility for local use 3 messages as defined in
       RFC 5424.";
   }

   identity local4 {
     base syslog-facility;
    description
      "The facility for local use 4 messages as defined in
       RFC 5424.";
   }

   identity local5 {
     base syslog-facility;
    description
      "The facility for local use 5 messages as defined in
       RFC 5424.";
   }

   identity local6 {
     base syslog-facility;
    description
      "The facility for local use 6 messages as defined in
       RFC 5424.";
   }

   identity local7 {
     base syslog-facility;
```

```
    description
      "The facility for local use 7 messages as defined in
       RFC 5424.";
  }
}

<CODE ENDS>
```

4.2. SYSLOG module

```
<CODE BEGINS> file "ietf-syslog.yang"

module ietf-syslog {
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog";
  prefix syslog;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-interfaces {
    prefix if;
  }

  import ietf-syslog-types {
    prefix syslogtypes;
  }

  organization "IETF NETMOD (NETCONF Data Modeling Language)
  Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

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

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

     Editor:   Clyde Wildes
               <mailto:cwildes@cisco.com>

     Editor:   Agrahara Kiran Koushik
               <mailto:kkoushik@brocade.com>";

  description
    "This module contains a collection of YANG definitions
     for Syslog configuration.";

  revision 2015-03-05 {
    description
      "Initial Revision";
    reference
      "This model references RFC 5424 - The Syslog Protocol,
       and RFC 5848 - Signed Syslog Messages.";
  }

  feature global-logging-action {
    description
      "This feature represents the ability to suppress log
       messages on the global level.";
  }
```

```
  feature file-logging-structured-data {
    description
      "This feature represents the ability to log messages
       to a file in structured-data format as per RFC 5424.";
  }

  feature remote-logging-structured-data {
    description
      "This feature represents the ability to deliver log
       messages to a remote server in structured-data format
       as per RFC 5424.";
  }

  feature file-logging-archive-config {
    description
      "This feature represents the ability to archive log files.";
  }

  feature remote-logging-use-vrf {
    description
      "This feature allows remote logging of messages to a
       particular VRF.";
  }

  feature terminal-facility-user-logging-config {
    description
      "This feature represents the ability to adjust
       log message settings for individual terminal users.";
  }

  feature selector-advanced-level-processing-config {
    description
      "This feature represents the ability to select messages
       using the additional operators equal to, or not equal to
       when comparing the Syslog message severity.";
  }

  feature selector-match-processing-config {
    description
      "This feature represents the ability to select messages based
       on a Posix 1003.2 regular expression pattern match.";
  }

  feature signed-messages-config {
    description
      "This feature represents the ability to configure signed
       syslog messages according to RFC 5848.";
  }

  grouping syslog-severity {
    description
      "This grouping defines the Syslog severity which is used to
       filter log messages.";
    leaf severity {
      type syslogtypes:severity;
      description
        "This leaf specifies the Syslog message severity.
         No value implies all severities.";
    }
  }
```

```
  grouping syslog-selector {
    description
      "This grouping defines a Syslog selector which is used to
       filter log messages for the given action in which the
       selector appears. Choose one of the following:
         logging-facility-all <severity>
         logging-facility-none
         logging-facility [<facility> <severity>...]
       Additional severity comparison operations are available
       using the logging-advanced-level-processing container. If
       the logging-advanced-level-processing container is not
       present all messages of the specified severity and higher
       are logged according to the given action.";
    choice logging-level-scope {
      default logging-facility-all;
      description
        "This choice describes the option to specify all
         facilities, no facilities, or a specific facility.";
      case logging-facility-all {
        description
          "This case specifies all facilities will match when
           comparing the Syslog message facility.";
        uses syslog-severity;
      }
      case logging-facility-none {
        description
          "This case specifies no facilities will match when
           comparing the Syslog message facility. This is a method
           that can be used to turn an action off.";
        leaf none {
          type empty;
          description
          "This leaf specifies that no facilities participate in the
           filtering of Syslog messages for this action.";
        }
      }
      case logging-facility {
        description
          "This case specifies one or more specified facilities
           will match when comparing the Syslog message facility.";
        list logging-facilities {
          key "facility";
          description
            "This list describes a collection of Syslog facilities
             and severities.";
          leaf facility {
            type identityref {
              base syslogtypes:syslog-facility;
            }
            description
              "The leaf uniquely identifies a Syslog facility.";
          }
          uses syslog-severity;
        }
      }
    }
```

```
    container logging-advanced-level-processing {
      if-feature selector-advanced-level-processing-config;
      description
        "This container describes the configuration parameters for
         advanced Syslog selector severity comparison.";
      leaf select-message-severity {
        type enumeration {
          enum equals-or-higher {
            description
              "All messages of the specified severity and higher are
               logged according to the given action";
          }
          enum equals {
            description
              "This leaf specifies all messages for the specified
               severity.";
          }
          enum not-equals {
            description
              "This leaf specifies all messages that are not for the
               specified severity.";
          }
        }
        default equals-or-higher;
        description
          "This leaf describes the option to specify how the
           severity comparison is performed.";
      }
    }
    container logging-match-processing {
      if-feature selector-match-processing-config;
      description
        "This container describes the configuration parameters for
         matching Syslog messages using a regular expression pattern
         match.";
      leaf pattern-match {
        type string;
        description
          "This leaf desribes a Posix 1003.2 regular expression
           string that can be used to select a Syslog message for
           logging. The match is performed on the RFC 5424
           SYSLOG-MSG field.";
      }
    }
  }

  container syslog {
    description
      "This container describes the configuration parameters for
       Syslog.";
    container global-logging-action {
      if-feature global-logging-action;
      description
        "This container describes the configuration parameters for
         global logging. Global logging represents the ability to
         perform global log message suppression.";
      uses syslog-selector;
    }
```

```
    container console-logging-action {
      description
        "This container describes the configuration parameters for
         console logging.";
      uses syslog-selector;
    }
    container buffered-logging-action {
      description
        "This container describes the configuration parameters for
         local memory buffer logging.";
      uses syslog-selector;
      leaf buffer-size {
        type uint64;
        description
          "This leaf describes the amount of memory that will be
           dedicated to local memory buffer logging. The default
           value varies by implementation.";
      }
    }
    container file-logging-action {
      description
        "This container describes the configuration parameters for
         file logging.";
      list logging-files {
        key "file-name";
        description
          "This list describes a collection of local logging
           files.";
        leaf file-name {
          type inet:uri;
          description
            "This leaf specifies the name of the log file.";
        }
        uses syslog-selector;
        leaf file-logging-structured-data {
          if-feature file-logging-structured-data;
          type boolean;
          default false;
          description
            "This leaf describes how log messages are written to the
             log file. If true, messages will be written in
             structured-data format; if false, messages will be
             written in standard message format.";
        }
        container file-logging-archive {
          if-feature file-logging-archive-config;
          description
            "This container describes the configuration parameters
             for log file archiving.";
          leaf file-number {
            type uint32;
            default 1;
            description
              "This leaf specifies the maximum number of log files
               retained.";
          }
```

```
          leaf file-size {
            type uint64;
            default 262144;
            description
              "This leaf specifies the maximum log file size.";
          }
          leaf file-permission {
            type enumeration {
               enum world-readable {
                 value 1;
                 description
                   "This enum specifies that the log files
                    are readable by world.";
               }
               enum no-world-readable {
                 value 2;
                 description
                   "This enum specifies that the log files
                    are not readable by world.";
               }
            }
            default no-world-readable;
            description
              "This leaf describes who can read log files";
          }
        }
      }
    }
    container remote-logging-action {
      description
        "This container describes the configuration parameters for
         remote logging.";
      list remote-logging-destination {
        key "destination";
        description
          "This list describes a collection of remote logging
           destinations.";
        leaf destination {
          type inet:host;
          description
            "The leaf uniquely specifies the address of the
             remote host. One of the following must be specified:
             an ipv4 address, an ipv6 address, or a host name.";
        }
        uses syslog-selector;
        leaf remote-logging-structured-data {
          if-feature remote-logging-structured-data;
          type boolean;
          default false;
          description
            "This leaf describes how log messages are sent to the
             remote server. If true, messages will be sent in
             structured-data format; if false, messages will be
             sent in standard message format.";
        }
        leaf destination-port {
          type inet:port-number;
          default 514;
          description
            "This leaf specifies the port number used to deliver
             messages to the remote server.";
        }
```

```
        leaf destination-facility {
          type identityref {
            base syslogtypes:syslog-facility;
          }
          default syslogtypes:local7;
          description
            "This leaf specifies the facility used in messages
             delivered to the remote server.";
        }
        leaf source-interface {
          type if:interface-ref;
          description
            "This leaf sets the source interface for the remote
             Syslog server. Either the interface name or the
             interface IP address can be specified. If not set,
             messages sent to a remote syslog server will contain
             the IP address of the interface the syslog message
             uses to exit the network element";
        }
        leaf vrf-name {
          if-feature remote-logging-use-vrf;
          type string;
          description
            "This leaf specifies the name of the virtual routing
             facility (VRF) that connects to the syslog server
             host. If not set, the default VRF will be used.";
        }
        container syslog-sign {
          if-feature signed-messages-config;
          presence
            "If present, syslog-sign is activated.";
          description
            "This container describes the configuration parameters
             for signed syslog messages as described by RFC 5848.";
          leaf cert-initial-repeat {
            type uint16;
            mandatory true;
            description
            "This leaf specifies the number of times each
             Certificate Block should be sent before the first
             message is sent.";
          }
          leaf cert-resend-delay {
            type uint16;
            mandatory true;
            description
              "This leaf specifies the maximum time delay in seconds
               until resending the Certificate Block.";
          }
          leaf cert-resend-count {
            type uint16;
            mandatory true;
            description
              "This leaf specifies the maximum number of other
               syslog messages to send until resending the
               Certificate Block.";
          }
          leaf sig-max-delay {
            type uint16;
            mandatory true;
            description
              "This leaf specifies when to generate a new Signature
               Block. If this many seconds have elapsed since the
               message with the first message number of the
```

```
                Signature Block was sent, a new Signature Block
                should be generated.";
          }
```

```
          leaf sig-number-resends {
            type uint16;
            mandatory true;
            description
              "This leaf specifies the number of times a Signature
               Block is resent. (It is recommended to select a value
               of greater than 0 in particular when the UDP
               transport [RFC5426] is used.).";
          }
          leaf sig-resend-delay {
            type uint16;
            mandatory true;
            description
              "This leaf specifies when to send the next Signature
               Block transmission based on time. If this many
               seconds have elapsed since the previous sending of
               this Signature Block, resend it.";
          }
          leaf sig-resend-count {
            type uint16;
            mandatory true;
            description
              "This leaf specifies when to send the next Signature
               Block transmission based on a count. If this many
               other syslog messages have been sent since the
               previous sending of this Signature Block, resend it.";
          }
        }
      }
    }
    container terminal-logging-action {
      description
        "This container describes the configuration parameters for
         the terminal logging configuration.";
      choice user-scope {
        default all-users;
        description
          "This choice describes the option to specify all users
           or a specific user. The all users case implies that
           messages will be sent to all terminals";
        case all-users {
          description
            "This case specifies all users.";
          container all-users {
            description
              "This container describes the configuration parameters
               for all users.";
            uses syslog-selector;
          }
        }
        case per-user {
          if-feature terminal-facility-user-logging-config;
          description
            "This case specifies a specific user.";
          list user-name {
            key "uname";
            description
              "This list describes a collection of user names.";
```

```
            leaf uname {
              type string;
              description
                "This leaf uniquely describes a user name which is
                 the login name of the user whose terminal session
                 is to receive log messages.";
            }
            uses syslog-selector;
          }
        }
      }
    }
  }
}

<CODE ENDS>
```

### 4.3. A SYSLOG Example

```
   Requirement:
   Enable global logging of two facilities:
     kern - severity critical(1)
     auth - severity error(3)

   Enable console logging of syslogs of severity
   critical(1)

   Here is the example syslog configuration xml:
   <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
       <target>
         <running/>
       </target>
       <config>
         <syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
           <global-logging-action>
             <logging-facilities>
               <facility>syslogtypes:kern</facility><severity>syslogtypes:critic
al</severity>
             </logging-facilities>
             <logging-facilities>
               <facility>syslogtypes:auth</facility><severity>syslogtypes:error<
/severity>
             </logging-facilities>
           </global-logging-action>
           <console-logging-action>
             <severity>syslogtypes:critical</severity>
           </console-logging-action>
         </syslog>
       </config>
     </edit-config>
   </rpc>

   <?xml version="1.0" encoding="UTF-8"?>
   <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ok/>
   </rpc-reply>
```

## 5. Implementation Status

[Note to RFC Editor: Please remove this section before publication.]

This section records the status of known implementations of the Syslog YANG model at the time of posting of this Internet-Draft.

Cisco Systems, Inc. has implemented the proposed IETF Syslog model for the Nexus 7000 NXOS OS as a prototype, together with an augmentation model for operating system specific Syslog configuration features.

Five leaves were implemented in the base IETF model and three leaves were implemented in the Cisco specific augmentation model as follows:

```
    Leaf XPATH                    Sample NXOS CLI Command(s)

syslog:global-logging-action   logging level cron 2
syslog:console-logging-action  logging console 1
syslog:file-logging-action     logging logfile mylog.log 2 4096
syslog:terminal-logging-action logging monitor 2
syslog:remote-logging-action  *logging server server.cisco.com 2
                                       facility user
                                       use-vrf management
                              *logging source-interface loopback 0
cisco-syslog:logging-timestamp-config  logging timestamp milli-seconds
cisco-syslog:origin-id-cfg  logging origin-id string abcdef
cisco-syslog:module-logging logging module 1
```

*The "logging server" and "logging source-interface" commands were combined into one base model leaf.

The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

## 6. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241] [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242] [RFC6242]. The NETCONF access control model [RFC6536] [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

TBD: List specific Subtrees and data nodes and their sensitivity/ vulnerability.

## 7. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688] [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:syslog

Registrant Contact: The IESG.

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

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: syslog namespace: urn:ietf:params:xml:ns:yang:syslog prefix: syslog reference: RFC XXXX

## 8. Acknowledgements

The authors wish to thank the following who commented on version 00 of this prop osal:

Jim Gibson <gibson@cisco.com>
Jeffrey Haas <jhaas@pfrc.org>
John Heasley <heas@shrubbery.net>
Giles Heron <giheron@cisco.com>
Lisa Huang <yihuan@cisco.com>
Jeffrey K Lange <jeffrey.K.lange@ge.com>
Chris Lonvick <lonvick@gmail.com>
Juergen Schoenwaelder <j.schoenwaelder@jacobs-university.de>
Peter Van Horne <petervh@cisco.com>
Bert Wijnen <bertietf@bwijnen.net>
Aleksandr Zhdankin <azhdanki@cisco.com>

## 9. Change log [RFC Editor: Please remove]

## 10. References

[RFC3164] Lonvick, C., "The BSD syslog Protocol", BCP 81, RFC 3164, August 2001.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, March 2204.

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

Authors' Addresses

Clyde Wildes
Cisco Systems Inc.

Email: cwildes@cisco.com

Kiran Agrahara Sreenivasa
Brocade Communications Systems

Email: kkoushik@brocade.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2015

L. Lhotka
CZ.NIC

February 24, 2015

# JSON Encoding of Data Modeled with YANG
# draft-ietf-netmod-yang-json-03

## Abstract

This document defines encoding rules for representing configuration, state data, RPC input and output parameters, and notifications defined using YANG as JavaScript Object Notation (JSON) text.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

The NETCONF protocol [RFC6241] uses XML [W3C.REC-xml-20081126] for encoding data in its Content Layer. Other management protocols might want to use other encodings while still benefiting from using YANG [RFC6020] as the data modeling language.

For example, the RESTCONF protocol [I-D.ietf-netconf-restconf] supports two encodings: XML (media type "application/yang.data+xml") and JSON (media type "application/yang.data+json).

The specification of the YANG data modelling language [RFC6020] defines only XML encoding for data instances, i.e. contents of configuration datastores, state data, RFC input and output parameters, and event notifications. The aim of this document is to define rules for encoding the same data as JavaScript Object Notation (JSON) text [RFC7159].

In order to achieve maximum interoperability while allowing implementations to use a variety of available JSON parsers, the JSON encoding rules follow, as much as possible, the constraints of the I-JSON restricted profile [I-D.ietf-json-i-json]. Section 7 discusses I-JSON conformance in more detail.

## 2. Terminology and Notation

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

The following terms are defined in [RFC6020]:

- o anyxml
- o augment
- o container
- o data node
- o identity
- o instance identifier
- o leaf
- o leaf-list
- o list
- o module
- o submodule

## 3. Validation of JSON-encoded Instance Data

Instance data validation as defined in [RFC6020] is only applicable to XML-encoded data. For one, semantic constraints in "must"

statements are expressed using XPath 1.0 [W3C.REC-xpath-19991116], which can be properly interpreted only in the XML context.

This document and the corresponding "XML Mapping Rules" sections from [RFC6020] also define an implicit schema-driven mapping of JSON-encoded instances to XML-encoded instances (and vice versa). This mapping is mostly straightforward. In cases where doubts could arise, this document gives explicit instructions for mapping JSON-encoded instances to XML.

In order to validate a JSON instance document, it MUST first be mapped, at least conceptually, to the corresponding XML instance document. By definition, the JSON document is then valid if and only if the XML document is valid according to the rules stated in [RFC6020].

## 4. Names and Namespaces

Instances of YANG data nodes (leafs, containers, leaf-lists, lists and anyxml nodes) are always encoded as members of a JSON object, i.e., as name/value pairs. This section defines how the name part is formed, and the following sections deal with the value part.

Except in the cases specified below, the member name is identical to the identifier of the corresponding YANG data node. Every such name belongs to a namespace which is associated with the YANG module where the corresponding data node is defined. If the data node is defined in a submodule, then the namespace is determined by the main module to which the submodule belongs.

If the namespace of a member name has to be explicitly specified, the module name SHALL be used as a prefix to the (local) member name. Both parts of the member name SHALL be separated with a colon character (":"). In other words, the namespace-qualified name will have the following form:

```
<module name>:<local name>
```

Figure 1: Encoding a namespace identifier with a local name.

Names with namespace identifiers in the form shown in Figure 1 are used if and only if the parent data node belongs to a different namespace, which also includes all top-level YANG data nodes.

For example, consider the following YANG module:

```
module foomod {

  namespace "http://example.com/foomod";

  prefix "foo";

  container top {
    leaf foo {
      type uint8;
    }
  }
}
```

If the data model consists only of this module, then the following is a valid JSON-encoded configuration:

```
{
  "foomod:top": {
    "foo": 54
  }
}
```

Note that the top-level container instance contains the namespace identifier (module name) but the "foo" leaf doesn’t because it is defined in the same module as its parent container.

Now, assume the container "top" is augmented from another module, "barmod":

```
module barmod {

  namespace "http://example.com/barmod";

  prefix "bar";

  import foomod {
    prefix "foo";
  }

  augment "/foo:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

A valid JSON-encoded configuration containing both leafs may then look like this:

```
{
  "foomod:top": {
    "foo": 54,
    "barmod:bar": true
  }
}
```

The name of the "bar" leaf is prefixed with the namespace identifier because its parent is defined in a different module, hence it belongs to another namespace.

Explicit namespace identifiers are sometimes needed when encoding values of the "identityref" and "instances-identifier" types. The same form as shown in Figure 1 is then used as well. See Sections 6.8 and 6.11 for details.

## 5. Encoding of YANG Data Node Instances

Every complete JSON instance document, such as a configuration datastore content, is an object. Its members are instances of all top-level data nodes defined by the YANG data model.

Character encoding MUST be UTF-8.

Any data node instance is encoded as a name/value pair where the name is formed from the data node identifier using the rules of Section 4. The value depends on the category of the data node as explained in the following subsections.

### 5.1. The "leaf" Data Node

A leaf instance is encoded as a name/value pair where the value can be a string, number, literal "true" or "false", or the special array "[null]", depending on the type of the leaf (see Section 6 for the type encoding rules).

Example: For the leaf node definition

```
leaf foo {
  type uint8;
}
```

the following is a valid JSON-encoded instance:

```
"foo": 123
```

## 5.2. The "container" Data Node

An container instance is encoded as a name/object pair. The container’s child data nodes are encoded as members of the object.

Example: For the container definition

```
container bar {
  leaf foo {
    type uint8;
  }
}
```

the following is a valid instance:

```
"bar": {
  "foo": 123
}
```

## 5.3. The "leaf-list" Data Node

A leaf-list is encoded as a name/array pair, and the array elements are values of the same type, which can be a string, number, literal "true" or "false", or the special array "[null]", depending on the type of the leaf-list (see Section 6 for the type encoding rules).

The ordering of array elements follows the same rules as the ordering of XML elements representing leaf-list entries in the XML encoding. Specifically, the "ordered-by" properties (sec. 7.7.5 in [RFC6020]) MUST be observed.

Example: For the leaf-list definition

```
leaf-list foo {
  type uint8;
}
```

the following is a valid instance:

```
"foo": [123, 0]
```

## 5.4. The "list" Data Node

A list instance is encoded as a name/array pair, and the array elements are JSON objects.

The ordering of array elements follows the same rules as the ordering of XML elements representing list entries in the XML encoding.

Specifically, the "ordered-by" properties (sec. 7.7.5 in [RFC6020]) MUST be observed.

Unlike the XML encoding, where list keys are required to precede any other siblings within a list entry, and appear in the order specified by the data model, the order of members in a JSON-encoded list entry is arbitrary because JSON objects are fundamentally unordered collections of members.

Example: For the list definition

```
list bar {
  key foo;
  leaf foo {
    type uint8;
  }
  leaf baz {
    type string;
  }
}
```

the following is a valid instance:

```
"bar": [
  {
    "foo": 123,
    "baz": "zig"
  },
  {
    "baz": "zag",
    "foo": 0
  }
]
```

## 5.5. The "anyxml" Data Node

An anyxml instance is encoded as a name/value pair. The value can be of any valid JSON type, i.e. an object, array, number, string or one of the literals "true", "false" and "null".

This document imposes no other restrictions on the contents of JSON-encoded anyxml instances. It also doesn’t define any universal mapping between the contents of JSON- and XML-encoded anyxml instances - note that such a mapping is not needed for the purposes of validation (Section 3) because anyxml contents are not subject to YANG-based validation (see sec. 7.10 in [RFC6020]). However, each definition of an anyxml node MAY specify, in its "description"

statement, appropriate syntactic, semantic and mapping rules for the values of that anyxml data node.

Example: For the anyxml definition

```
anyxml bar;
```

the following is a valid instance:

```
"bar": [true, null, true]
```

## 6. The Mapping of YANG Data Types to JSON Values

The type of the JSON value in an instance of the leaf or leaf-list data node depends on the type of that data node as specified in the following subsections.

### 6.1. Numeric Types

A value of the "int8", "int16", "int32", "uint8", "uint16" and "uint32" is represented as a JSON number.

A value of the "int64", "uint64" or "decimal64" type is encoded as a JSON string whose contents is the lexical representation of that numeric value as specified in sections 9.2.1 and 9.3.1 of [RFC6020].

For example, if the type of the leaf "foo" in Section 5.1 was "uint64" instead of "uint8", the instance would have to be encoded as

```
"foo": "123"
```

The special handling of 64-bit numbers follows from I-JSON recommendation to encode numbers exceeding the IEEE 754-2008 double precision range as strings, see sec. 2.2 in [I-D.ietf-json-i-json].

### 6.2. The "string" Type

A "string" value encoded as a JSON string, subject to JSON string encoding rules.

### 6.3. The "boolean" Type

A "boolean" value is mapped to the corresponding JSON literal name "true" or "false".

### 6.4. The "enumeration" Type

An "enumeration" value is mapped in the same way as a string except that the permitted values are defined by "enum" statements in YANG. See sec. 9.6 in [RFC6020].

### 6.5. The "bits" Type

A "bits" value is mapped to a JSON string identical to the lexical representation of this value in XML, i.e., space-separated names representing the individual bit values that are set. See sec. 9.7 in [RFC6020].

### 6.6. The "binary" Type

A "binary" value is mapped to a JSON string identical to the lexical representation of this value in XML, i.e., base64-encoded binary data. See sec. 9.8 in [RFC6020].

### 6.7. The "leafref" Type

A "leafref" value is mapped according to the same rules as the type of the leaf being referred to.

### 6.8. The "identityref" Type

An "identityref" value is mapped to a string representing the name of an identity. Its namespace MUST be expressed as shown in Figure 1 if it is different from the namespace of the leaf node containing the identityref value, and MAY be expressed otherwise.

For example, consider the following schematic module:

```
module exmod {
  ...
  import ietf-interfaces {
    prefix if;
  }
  import iana-if-type {
    prefix ianaift;
  }
  ...
  leaf type {
    type identityref {
      base "if:interface-type";
    }
  }
}
```

A valid instance of the "type" leaf is then encoded as follows:

```
"type": "iana-if-type:ethernetCsmacd"
```

The namespace identifier "iana-if-type" must be present in this case because the "ethernetCsmacd" identity is not defined in the same module as the "type" leaf.

### 6.9. The "empty" Type

An "empty" value is mapped to "[null]", i.e., an array with the "null" literal being its only element.

This encoding was chosen instead of using simply "null" in order to facilitate the use of empty leafs in common programming languages. When used in a boolean context, the "[null]" value, unlike "null", evaluates to true.

Example: For the leaf definition

```
leaf foo {
  type empty;
}
```

a valid instance is

```
"foo": [null]
```

### 6.10. The "union" Type

A value of the "union" type is encoded as the value of any of the member types.

Unlike XML, JSON conveys part of the type information already in the encoding. When validating a value of the "union" type, this information MUST also be taken into account.

For example, consider the following YANG definition:

```
leaf bar {
  type union {
    type uint16;
    type string;
  }
}
```

In RESTCONF [I-D.ietf-netconf-restconf], it is fully acceptable to set the value of "bar" in the following way when using the "application/yang.data+xml" media type:

```
<bar>13.5</bar>
```

because the value may be interpreted as a string, i.e., the second member type of the union. When using the "application/yang.data+json" media type, however, this is an error:

```
"bar": 13.5
```

In this case, the JSON encoding indicates the value is supposed to be a number rather than a string.

### 6.11. The "instance-identifier" Type

An "instance-identifier" value is encoded as a string that is analogical to the lexical representation in XML encoding, see sec. 9.13.3 in [RFC6020]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 4, namely:

- o The namespace identifier is the module name where each data node is defined.
- o The encoding of a node name with an explicit namespace is as shown in Figure 1.
- o The leftmost (top-level) node name is always prefixed with the namespace identifier.
- o Any subsequent node name has the namespace identifier if and only if its parent node has a different namespace. This also holds for node names appearing in predicates.

For example,

```
/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip
```

is a valid instance-identifer value because the data nodes "interfaces", "interface" and "name" are defined in the module "ietf-interfaces", whereas "ipv4" and "ip" are defined in "ietf-ip".

When translating an instance-identifier value from JSON to XML, the namespace identifier (YANG module name) in each component of the instance-identifier MUST be replaced by an XML namespace prefix that

is associated with the namespace URI reference of the module in the scope of the element containing the instance-identifier value.

## 7. I-JSON Compliance

I-JSON [I-D.ietf-json-i-json] is a restricted profile of JSON that guarantees maximum interoperability for protocols that use JSON in their messages, no matter what JSON encoders/decoders are used in protocol implementations. The encoding defined in this document therefore observes the I-JSON requirements and recommendations as closely as possible.

In particular, the following properties are guaranteed:

- o Character encoding is UTF-8.
- o Member names within the same JSON object are always unique.
- o The order of JSON object members is never relied upon.
- o Numbers of any type supported by YANG can be exchanged reliably. See Section 6.1 for details.

The only two cases where a JSON instance document encoded according to this document may deviate from I-JSON were dictated by the need to be able to encode the same instance data in both JSON and XML. These two exceptions are:

- o Leaf values encoded as strings may contain code points identifying Noncharacters that belong to the XML character set (see sec. 2.2 in [W3C.REC-xml-20081126]). This issue is likely to be solved in YANG 1.1 because noncharacters will not be allowed in string values, see sec. 9.4 in [I-D.ietf-netmod-rfc6020bis].
- o Values of the "binary" type are encoded with the base64 encoding scheme (Section 6.6), whereas I-JSON recommends base64url instead. Theoretically, values of the "binary" type might appear in URI references, such as Request-URI in RESTCONF, although in practice the cases where it is really needed should be extremely rare.

## 8. Security Considerations

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, it doesn’t contribute any new security issues beyond those discussed in sec. 15 of [RFC6020].

JSON is rather different from XML, and JSON parsers may thus suffer from other types of vulnerabilities than their XML counterparts. To

minimize these security risks, it is important that client and server software supporting JSON encoding behaves as required in sec. 3 of [I-D.ietf-json-i-json]. That is, received JSON data that violate any of I-JSON strict constraints MUST NOT be trusted nor acted upon. Violations due to the presence of Unicode Noncharacters in the data (see Section 7) SHOULD be carefully examined.

## 9. Acknowledgments

The author wishes to thank Andy Bierman, Martin Bjorklund, Dean Bogdanovic, Balazs Lengyel, Juergen Schoenwaelder and Phil Shafer for their helpful comments and suggestions.

## 10. References

### 10.1. Normative References

[I-D.ietf-json-i-json]
Bray, T., "The I-JSON Message Format", draft-ietf-json-i-json-06 (work in progress), January 2015.

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

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

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

### 10.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", draft-ietf-netmod-rfc6020bis-03 (work in progress), January 2015.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[W3C.REC-xpath-19991116]
Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

## Appendix A. A Complete Example

The JSON document shown below represents the same data as the reply to the NETCONF <get> request appearing in Appendix D of [RFC7223]. The data model is a combination of two YANG modules: "ietf-interfaces" and "ex-vlan" (the latter is an example module from Appendix C of [RFC7223]). The "if-mib" feature defined in the "ietf-interfaces" module is considered to be active.

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ex-vlan:vlan-tagging": true
      },
      {
        "name": "eth1.10",
        "type": "iana-if-type:l2vlan",
        "enabled": true,
        "ex-vlan:base-interface": "eth1",
        "ex-vlan:vlan-id": 10
      },
      {
        "name": "lo1",
        "type": "iana-if-type:softwareLoopback",
        "enabled": true
```

```
        }
      ]
    },
    "ietf-interfaces:interfaces-state": {
      "interface": [
        {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "admin-status": "down",
          "oper-status": "down",
          "if-index": 2,
          "phys-address": "00:01:02:03:04:05",
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "admin-status": "up",
          "oper-status": "up",
          "if-index": 7,
          "phys-address": "00:01:02:03:04:06",
          "higher-layer-if": [
            "eth1.10"
          ],
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "eth1.10",
          "type": "iana-if-type:l2vlan",
          "admin-status": "up",
          "oper-status": "up",
          "if-index": 9,
          "lower-layer-if": [
            "eth1"
          ],
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "eth2",
          "type": "iana-if-type:ethernetCsmacd",
          "admin-status": "down",
          "oper-status": "down",
```

```
            "if-index": 8,
            "phys-address": "00:01:02:03:04:07",
            "statistics": {
              "discontinuity-time": "2013-04-01T03:00:00+00:00"
            }
          },
          {
            "name": "lo1",
            "type": "iana-if-type:softwareLoopback",
            "admin-status": "up",
            "oper-status": "up",
            "if-index": 1,
            "statistics": {
              "discontinuity-time": "2013-04-01T03:00:00+00:00"
            }
          }
        ]
      }
    }
```

## Appendix B. Change Log

RFC Editor: Remove this section upon publication as an RFC.

### B.1. Changes Between Revisions -02 and -03

- o Namespace encoding is defined without using RFC 2119 keywords.
- o Specification for anyxml nodes was extended and clarified.
- o Text about ordering of list entries was corrected.

### B.2. Changes Between Revisions -01 and -02

- o Encoding of namespaces in instance-identifiers was changed.
- o Text specifying the order of array elements in leaf-list and list instances was added.

### B.3. Changes Between Revisions -00 and -01

- o Metadata encoding was moved to a separate I-D, draft-lhotka-netmod-yang-metadata.
- o JSON encoding is now defined directly rather than via XML-JSON mapping.

- o The rules for namespace encoding has changed. This affect both node instance names and instance-identifiers.
- o I-JSON-related changes. The most significant is the string encoding of 64-bit numbers.
- o When validating union type, the partial type info present in JSON encoding is taken into account.
- o Added section about I-JSON compliance.
- o Updated the example in appendix.
- o Wrote Security Considerations.
- o Removed IANA Considerations as there are none.

## Author’s Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2015

L. Lhotka
CZ.NIC

February 28, 2015

# Defining and Using Metadata with YANG
# draft-lhotka-netmod-yang-metadata-01

## Abstract

This document defines a YANG extension statement that allows for defining the syntax of metadata annotions in YANG modules. The document also specifies the XML and JSON encoding of annotations and other rules for annotating instances of YANG data nodes.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

There is a need to be able to annotate instances of YANG [3] data nodes with metadata. Typical use cases are:

- o Deactivating a subtree in a configuration datastore while keeping the data in place.

- o Complementing data model information with instance-specific data.

- o RPC operations may use metadata annotations for various purposes in both requests and responses. For example, the <edit-config> operation in the NETCONF protocol (see section 7.2 of [5]) uses annotations in the form of XML attributes for identifying the point in the configuration and type of the operation.

However, metadata annotations could potentially lead to interoperability problems if they are used in an ad hoc way by different organizations and/or without proper documentation. A sound metadata framework for YANG should therefore satisfy these requirements:

1. The set of annotations must be extensible in a distributed manner so as to allow for defining new annotations without running into

the risk of collisions with annotations defined and used by others.

2. Syntax and semantics of annotations must be documented and the documentation must be easily accessible.

3. Clients of network management protocols such as NETCONF [5] or RESTCONF [10] must be able to learn all annotations supported by a given server and identify each of them correctly.

This document proposes a systematic way for defining the syntax of metadata annotations. For this purpose, YANG extension statement "annotation" is defined in the module "ietf-yang-metadata" (Section 6). Other YANG modules importing this module can use the "annotation" statement for defining the syntax one or more annotations.

Semantics of metadata annotations MUST be defined separately. How it is done is outside the scope of this document.

The benefits of defining the syntax of metadata annotations in a YANG module are the following:

- Each annotation is bound to a YANG module name, namespace URI and prefix. This makes its encoding in instance documents (both XML and JSON) straightforward and consistent with the encoding of YANG data node instances.

- Annotations are indirectly registered through IANA YANG module registration.

- Annotations are included in the data model. Specifically, servers indicate syntactic support for certain annotations using standard module advertisement methods, such as the <hello> message in NETCONF.

- Values of annotations are not limited to strings; any YANG built-in or derived type may be used for them.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

The following terms are defined in [5]:

- client,

- o datastore,
- o message,
- o operation,
- o server.

The following terms are defined in [3]:

- o anyxml,
- o built-in type,
- o derived type,
- o container,
- o data model,
- o data node,
- o derived type,
- o extension,
- o leaf-list,
- o list,
- o module,
- o RPC operation,
- o submodule,
- o type.

The following terms are defined in [8]:

- o attribute,
- o document,
- o element,
- o namespace,

- o prefix.

The following terms are defined in [6]:

- o array,
- o member,
- o object,
- o primitive type.

In the following text, XML element names and YANG extension statements are always written with explicit namespace prefixes that are assumed to be bound to URI references as shown in Table 1.

| Prefix | URI Reference |
|---|---|
| rng | http://relaxng.org/ns/structure/1.0 |
| md | urn:ietf:params:xml:ns:yang:ietf-yang-metadata |
| ein | http://example.org/example-inactive |

Table 1: Used namespace prefixes and corresponding URI references

## 3. Defining Annotations in YANG

Metadata annotations are defined with YANG extension statement "md:annotation". This YANG language extension is defined in the module "ietf-yang-metadata" (Section 6).

Substatements of "md:annotation" are shown in Table 2. They are all core YANG statements, and the numbers in the second column refer to the corresponding sections in RFC 6020 [3] where each statement is described.

| substatement | RFC 6020 section | cardinality |
|---|---|---|
| description | 7.19.3 | 0..1 |
| if-feature | 7.18.2 | 0..n |
| reference | 7.19.4 | 0..1 |
| status | 7.19.2 | 0..1 |
| type | 7.6.3 | 0..1 |
| units | 7.3.3 | 0..1 |

Table 2: Substatements of "md:annotation".

Using the "type" statement, a type may be specified for the annotation value according to the same rules as for YANG leaf or leaf-list types. However, the "type" statement is optional as a substatement of "md:annotation" statement. If it is not present, the built-in "string" type is the default.

The "if-feature" statement, if present, makes the annotation conditional: it is supported only by servers that advertise the corresponding feature.

For example, the following module defines the "inactive" annotation:

```
module example-inactive {
  namespace "http://example.org/example-inactive";
  prefix "ein";
  import ietf-yang-metadata {
    prefix "md";
  }
  md:annotation inactive {
    type boolean;
    description
      "If this annotation is attached to a configuration data node,
       then the data subtree rooted at this node is deactivated.";
  }
}
```

By advertising a YANG module in which metadata annotation X is defined using the "md:annotation" statement, a server specifies just the syntax of annotation X. This means that configuration or state data, RPC messages and notifications will be considered syntactically valid if annotation X is attached to any data node instance, provided that all rules stated in this document are observed. However, the semantics of an annotation including the expected behavior of clients and servers MUST be specified by other means that are outside the scope of this document.

## 4. The Encoding of Annotations

XML attributes are a natural choice for encoding metadata in XML instance documents. For JSON [6], there is no generally established method for encoding metadata. This document thus introduces a special encoding method that is consistent with the JSON encoding of YANG data node instances as defined in [7].

### 4.1. XML Encoding

Metadata annotations are added to XML-encoded instances of YANG data nodes as XML attributes according to these rules:

- o The local name of the attribute SHALL be the same as the name of the annotation specified in the argument of the corresponding "md:annotation" statement.

- o The namespace of the attribute SHALL be identified by the URI that appears as the argument of the "namespace" statement in the YANG module where the annotation is defined. It is RECOMMENDED that the prefix specified by the "prefix" statement in the same module is used in the qualified name of the attribute.

- o The attribute value SHALL be encoded in the same way as the value of a YANG leaf instance having the same type.

For example, the "inactive" annotation defined in Section 3 may be encoded as follows:

```
<foo xmlns:ein="http://example.org/example-inactive"
     ein:inactive="true">
  ...
</foo>
```

### 4.2. JSON Encoding

The JSON metadata encoding defined in this section has the following properties:

1. The encoding of YANG data node instances as defined in [7] does not change.

2. Namespaces of metadata annotations are encoded in the same way as namespaces of YANG data node instances, see [7].

#### 4.2.1. Metadata Object and Annotations

All metadata annotations assigned to a YANG data node instance are encoded as members (name/value pairs) of a single JSON object, henceforth denoted as the metadata object. The placement and name of this object depends on the type of the data node as specified in the following subsections.

The name of a metadata annotation (as a member of the metadata object) SHALL be of the following form:

```
MODULE_NAME:LOCAL_NAME
```

where MODULE_NAME is the name of the YANG module in which the annotation is defined, and LOCAL_NAME is the name of the annotation specified in the argument of the corresponding "md:annotation" statement.

Note that unlike YANG data node instances, for annotations the explicit namespace identifier (MODULE_NAME) must always be used.

The value of a metadata annotation SHALL be encoded in exactly the same way as the value of a YANG leaf node having the same type as the annotation.

#### 4.2.2. Adding Annotations to Container and List Instances

For an instance that is encoded as a JSON object (i.e., a container or list entry), the metadata object is added as a new member of that object with the name "@".

Examples:

o "cask" is a container node:

```
"cask": {
  "@": {
    "example-inactive:inactive": true
  },
  ...
}
```

o "seq" is a list whose key is "name", annotation "inactive" is added only to the first entry:

```
"seq": [
  {
    "@": {
      "example-inactive:inactive": true
    },
    "name": "one",
    ...
  },
  {
    "name": "two",
    ...
  }
]
```

### 4.2.3. Adding Annotations to Leaf or Anyxml Instances

For a leaf or anyxml instance, the metadata object is added as a sibling name/value pair whose the name is the symbol "@" concatenated with the name of the leaf or anyxml member that is being annotated.

For example, if "flag" is a leaf node of the "boolean" type:

```
"flag": true,
"@flag": {
  "example-inactive:inactive": true
}
```

### 4.2.4. Adding Annotations to Leaf-list Instances

For a leaf-list instance, which is represented as a JSON array with values of a primitive type, annotations may be assigned to one or more entries by adding a name/array pair as a sibling the leaf-list instance, where the name is the symbol "@" concatenated with the name of the leaf-list that is being annotated, and the value is a JSON array whose i-th element is the metadata object with annotations assigned to the i-th entry of the leaf-list instance, or null if the i-th entry has no annotations.

Trailing null values in the array, i.e., those following the last non-null metadata object, MAY be omitted.

For example, in the following leaf-list instance with four entries, the "inactive" annotation is added to the second and third entry in the following way:

```
"bibliomod:folio": [6, 3, 7, 8],
"@bibliomod:folio": [
  null,
  {"example-inactive:inactive": true},
  {"example-inactive:inactive": true}
]
```

## 5. Representing Annotations in DSDL Schemas

RFC 6110 [4] defines a standard mapping of YANG data models to Document Schema Definition Languages (DSDL) [9]. This section specifies the mapping for the extension statement "md:annotation" (Section 6), which enables validation of XML instance documents containing metadata annotations.

The first step of the DSDL mapping procedure, i.e., the transformation of the YANG data model to the hybrid schema (see sec. 6 in [4]), is modified as follows:

1. If the data model contains at least one "md:annotation" statement, then a RELAX NG named pattern definition MUST be added as a child of the root <rng:grammar> element in the hybrid schema. It is RECOMMENDED to use the name "__yang_metadata__" for this named pattern.

2. A reference to the named pattern described in item 1 MUST be included as a child of every <rng:element> pattern that corresponds to a container, leaf, list or leaf-list data node.

3. Every metadata annotation definition in the form

   ```
   md:annotation ARGUMENT;
   ```

   or

   ```
   md:annotation ARGUMENT {
     ...
   }
   ```

   is mapped to the following RELAX NG pattern:

   ```
   <rng:attribute name="PREFIX:ARGUMENT">
     ...
   </rng:attribute>
   ```

   where PREFIX is the namespace prefix bound to the namespace URI of the YANG module that contains the "md:annotation" statement. Each "rng:attribute" pattern SHALL be wrapped in the

<rng:optional> pattern, and this SHALL be inserted as a child of the named pattern definition described in item 1.

4. Substatements of "md:annotation", if there are any, SHALL be mapped to children of the "rng:attribute" pattern exactly as described in sec. 10 of [4].

For example, the named pattern definition (item 1), when constructed only for the "inactive" annotation, will have the following form:

```
<rng:define name="__yang_metadata__">
  <rng:optional>
    <rng:attribute name="ein:inactive">
      <rng:choice>
        <rng:value>true</rng:value>
        <rng:value>false</rng:value>
      </rng:choice>
    </rng:attribute>
  </rng:optional>
</rng:define>
```

Every "rng:element" pattern that corresponds to a container, leaf, list or leaf-list data node will then contain a reference to the above named pattern, for example

```
<rng:element name="foo:bar">
  <rng:ref name="__yang_metadata__"/>
  ...
</rng:element>
```

Note that it is not necessary to use such a reference for "rng:element" patterns corresponding to anyxml data nodes because they already permit any XML attributes to be attached to their instances.

The second step of the DSDL mapping procedure, i.e., the transformation of the hybrid schema to RELAX NG, Schematron and DSRL schemas, is unaffected by the inclusion of "md:annotation".

## 6. Metadata YANG Module

RFC Editor: In this section, replace all occurrences of ’XXXX’ with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-yang-metadata@2015-02-28.yang"

```
module ietf-yang-metadata {
```

```
     namespace "urn:ietf:params:xml:ns:yang:ietf-yang-metadata";

     prefix "md";

     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: Thomas Nadeau
                  <mailto:tnadeau@lucidvision.com>

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

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>";

     description
       "This YANG module defines an extension statement that allows for
        defining metadata annotations.

        Copyright (c) 2014 IETF Trust and the persons identified as
        authors of the code. All rights reserved.

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

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

     revision 2015-02-28 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: Defining and Using Metadata with YANG";
     }

     extension annotation {
       argument name;
       description
         "This extension allows for defining metadata annotations in
```

```
          YANG modules. The 'md:annotation' statement can appear only at
          the top level of a YANG module.

          An annotation defined with this extension statement inherits
          the namespace and other context from the YANG module in which
          it is defined.

          Other properties of the annotation and documentation may be
          specified using the following standard YANG substatements (all
          are optional): 'description', 'if-feature', 'reference',
          'status', 'type' and 'units'. If the 'type' statement is not
          present, the built-in 'string' type is used by default.

          A server announces syntactic support for a particular
          annotation by including the module in which the annotation is
          defined among the advertised YANG modules (e.g. in NETCONF
          hello message).

          The 'description' and/or 'reference' statements should provide
          links to the specification of the annotation's semantics.

          XML and JSON encoding of annotations is defined in
          RFC XXXX.";
    }
  }

<CODE ENDS>
```

## 7. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URI in the IETF XML registry [2]:

```
-----------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-yang-metadata

Registrant Contact: The IESG.

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

This document registers the following YANG module in the YANG Module Names registry [3]:

```
------------------------------------------------------------------
name:         ietf-yang-metadata
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-metadata
prefix:       md
reference:    RFC XXXX
------------------------------------------------------------------
```

## 8. Security Considerations

This document introduces a mechanism for defining the syntax of metadata annotations in YANG modules and using them with instances of YANG data nodes. By itself, this mechanism represents no security threat. Security implications of a particular annotation defined using this mechanism have to be duly considered and documented in the specification of the annotation’s semantics.

## 9. References

### 9.1. Normative References

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

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

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

[4] Lhotka, L., "Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content", RFC 6110, February 2011.

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

[6] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[7] Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-03 (work in progress), February 2015.

[8] Cowan, J. and R. Tobin, "XML Information Set (Second Edition)", World Wide Web Consortium Recommendation REC-xml-infoset-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-infoset-20040204>.

## 9.2. Informative References

[9] International Organization for Standardization, "Document Schema Definition Languages (DSDL) - Part 1: Overview", ISO/IEC 19757-1, November 2004.

[10] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

# Appendix A. Change Log

RFC Editor: Remove this section upon publication as an RFC.

## A.1. Changes Between Revisions -00 and -01

- o Encoding of annotations for anyxml nodes was changed to be the same as for leafs. This was necessary because anyxml value now needn’t be an object.

- o It is stated that "md:annotation" statement defines only the syntax of an annotation.

- o Allowed "if-feature" as a substatement of "md:annotation".

# Author’s Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

A. Shaikh
Google
R. Shakir
BT
K. D'Souza
AT&T
L. Fang
Microsoft

March 9, 2015

# Operational Structure and Organization of YANG Models
draft-openconfig-netmod-model-structure-00

## Abstract

This document presents an approach for organizing YANG models in a comprehensive structure that defines how individual models may be composed to configure and operate network infrastructure and services. The structure is itself represented as a YANG model rooted at a device, with all of the related component models logically organized in a way that is operationally intuitive.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.

# 1. Introduction

The large number of configuration models recently published cover much of networking protocols and technology and, in theory, enable a programmatic, model-driven approach for configuring network devices. These models have been largely developed individually and in isolation, however, making it challenging to use them together to fully configure a device, or manage a set of devices comprising a service. For example, standard models for interface management [RFC7223] and system management [RFC7317] are available but there is no guidance for how they should be used together, or combined with other models for routing protocols, ACLs, etc. to form a complete model. Recently, some frameworks (e.g., [RTG-CFG] and [RTG-POLICY]) that tie models together have been developed, but they are incomplete, covering only a subset of related models.

## 1.1. Goals and approach

In this document, we describe a structure for organizing YANG [RFC6020] models that is broadly applicable to physical and virtual devices. Individual models are composed such that the data they define can be accessed in a predictable and operationally intuitive way that is common across implementations. This organization enables several important capabilities:

- o a common schema to access data related to all aspects of a device

- o an extensible structure that makes it clear where additional models or data should be fit (e.g., using YANG augmentation or imports)

- o a place for including metadata that provides useful information about the corresponding individual models, such as which organization provides them, which vendors support them, or which version of the model is deployed

- o a common infrastructure model layer on which higher layer service models can be built, for example by specifying which models are needed to provide the service

- o an ability to express an instance of the structure consisting of models that have been validated to work together (i.e., with information about sources of the models, their versions, etc.), so that operators can easily identify a set of models that is known to be mutually consistent

Our approach is to organize the models describing various aspects of network infrastructure, including devices and their subsystems, and relevant protocols operating at the link and network layers. The proposal does not consider a common model for higher level network services, nor does it specify details of how hardware-related data should be organized. Both of these are challenging to standardize -- services are subject to operational and business considerations that vary across network operators, and hardware models are necessarily dependent on specific platform features and architecture -- and are thus out of scope of this document. We instead consider the set of models that are commonly used by network operators, and suggest a corresponding organization.

As with other models developed from an operator perspective, the intent is not to be exhaustive by including all possible models in the overall structure, whether currently available or not. We focus on components that are deemed most useful for network operators across a variety of use cases. We recognize, however, that additional models will be needed in some cases, and this structure is useful for describing how new models can be fit into the overall structure.

## 2. Model overview

The model organization can itself be thought of as a "meta- model", in that it describes the relationships between individual models. We choose to represent it also as simple YANG model consisting of lists and containers to serve as anchor points for the corresponding individual models.

As shown below, our model is rooted at a "device", which represents a network router, switch, or similar device. The model is applicable to both physical, hardware-based devices, as well as software-based devices such as virtual network functions (VNFs). It does not follow the hierarchy of any particular implementation, and hence is vendor-neutral. Nevertheless, the structure should be familiar to network operators and also readily mapped to vendor implementations.

```
+--rw device
      +--rw info
      |  +--rw device-type?
      |     ...
      +--rw hardware
      +--rw system
      |     ...
      +--rw interfaces
      |     ...
      +--rw acl
      +--rw qos
      +--rw logical-routers
            ...
```

The key subsystems are represented at the top level of the device, including, system-wide configuration, interfaces, and routing instances. The info section can be used for basic device information such as its type (e.g., physical or virtual), vendor, and model. For physical devices, the hardware container is intended to be a placeholder for platform-specific configuration and operational state data. For example, a common structure for the hardware model might include chassis, linecards, and ports, but we leave this unspecified.

## 2.1. System model components

The system container includes a number of subsystems that are typically configured globally for the device. Some of these, such as DHCP, Ethernet CFM, or sampling configuration also may have data that is associated with an interface. For simplicity, these relationships are not represented in this structural model. The currently defined subsystems are shown below:

```
+--rw device
   +--rw system
      +--rw dns
      +--rw ntp
      +--rw dhcp
      +--rw syslog
      +--rw ssh
      +--rw stat-coll
      +--rw oam
      |  +--rw snmp
      |  +--rw cfm
      |  +--rw twamp
      +--rw aaa
      |  +--rw tacacs
      |  +--rw radius
      +--rw users
```

## 2.2. Interface model components

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interfaces container includes a number of commonly used components as examples:

```
+--rw device
   +--rw interfaces
      +--rw ethernet
      |  +--rw aggregates
      |  +--rw vlans
      |  +--rw lfm
      +--rw sonet-sdh
      +--rw addressing
      |  +--rw ipv4
      |  |  +--rw vrrp
      |  +--rw ipv6
      |     +--rw vrrp
      +--rw tunnels
```

## 2.3. Logical routing instances

Logical routers represent the capability on some devices to partition resources into independent logical routers. In physical devices, some hardware features are shared across partitions, but routing protocol instances, routing tables, and configuration are managed separately. In virtual routers or VNFs, this may correspond to establishing multiple logical instances using a single software installation. The model supports configuration of multiple routing instances on a single device by creating a list of logical routers, each with their own configuration and operational state related to routing and switching protocols, as shown below:

```
+--rw device
   +--rw logical-routers
      +--rw logical-router* [router-id]
         +--rw router-id           uint8
         +--rw router-name?        string
         +--rw layer-2-protocols
         |     ...
         +--rw layer-3-protocols
               ...
```

### 2.4. VRFs and global routing configuration

Virtual routing and forwarding instances (VRFs) are commonly used to isolate routing domains, for example to create virtual private networks, each with their own active protocols and routing policies. Devices also have a global instance of each routing protocol that may also exchange routes with VRFs through routing policies. The model describes protocols and policies for both VRF routing instances and the global instance. The routing policy framework is expected to follow [RTG-POLICY], which enables import / export policies to be expressed with respect to a VRF, or the global routing instance.

```
+--rw device
      +--rw logical-routers
         +--rw logical-router* [router-id]
            +--rw router-id
            +--rw router-name?
            +--rw layer-3-protocols
               +--rw global
               |     ...
               +--rw vrf* [vrf-name]
               |     ...
               +--rw routing-policy
                     ...
```

## 3. Populating the structural model

The structural model in this document describes how individual YANG models may be used together to represent the configuration and operational state for all parts of a physical or virtual device. It does not, however, document the actual model in its entirety. In this section, we outline an option for creating the full model and also describe how it may be used.

### 3.1. Constructing the device model

One of the challenges in assembling existing YANG models is that they are generally written with the assumption that each model is at the root of the configuration or state tree. Combining models then results in a multi-rooted tree that does not follow any logical construction and makes it difficult to work with operationally. In some cases, models explicitly reference other models (e.g., via augmentation) to define a relationship, but this is the case for only a few existing models.

Some examples include the interfaces [RFC7223] and IP management [RFC7277] models, and proposed IS-IS [RTG-ISIS], OSPF [RTG-OSPF] and routing configuration [RTG-CFG] models.

### 3.2. Pull approach for model composition

To enable model composition, one possible approach is to avoid using root-level containers in individual component models. Instead, the top level container (and all other data definitions) can be enclosed in a YANG ’grouping’ statement so that when the model is imported by another model, its location in the configuration tree can be controlled by the importing YANG module with the ’uses’ statement. One advantage of this approach is that the importing module has the flexibility to readily use the data definitions where the author deems appropriate.

One obvious drawback is that individual models no longer contain any of their own data definitions and must be used by a higher-level model for their data nodes to become active. Some judgment as to which models are more suited for inclusion in higher level models is also necessary to decide when the corresponding YANG module should contain only groupings. Another potential drawback is that this approach does not define a common structure for models to fit together, limiting interoperability due to implementations using different structures. To address this, a top-level standard model structure could be defined and updated to import new models into the hierarchy as they are defined.

### 3.3. "Push" approach for model composition

An alternative approach is to develop a top level model which defines the overall structure of the models, similar to the structure described in Section 2. Individual models may augment the top level model with their data nodes in the appropriate locations. The drawback is the need for a pre-defined top level model structure. On the other hand, when this top level model is standardized, it can become the basis for a vendor-neutral way to manage devices, assuming that the component models are supported by a given implementation.

One question in both approaches is what the root of the top- level model should be. In this document we selected to base the mode at a device because this layer should be common across many use cases and implementations. Starting at a higher layer (e.g., services) makes defining and agreeing on a common organization more challenging as discussed in Section 1.1.

Ideally, one could consider a hybrid construction mechanism that supports both styles of model composition. For example, a YANG compiler directive could be used to indicate whether an individual model should assume it is at the root, or whether it is meant for inclusion in other higher-level models.

## 4. Additional use cases

The goal of this document is to motivate the need for an overall structure for YANG data models that allows all of the data to be accessed in a common, logical way. With such a structure defined itself as a simple YANG model, it is possible to consider additional use cases.

### 4.1. Model catalog

YANG data models are being developed in a number of organizations, including standards bodies such as IETF, ONF, and IEEE, as well as open source projects and ad-hoc working groups. In addition to understanding how these models can work together, another challenge for users is the complexity of tracking which organization created a given model, and the capabilities and coverage each model provides. This becomes even more difficult when multiple overlapping models are available for a particular component.

Such a catalog could also be locally defined by an operator to describe the models needed to instantiate and manage different services.

The idea of a model catalog is similar to service catalogs in traditional IT environments. Service catalogs serve as a software-based registries of available services with information needed to discover and invoke available services.

The current model structure described in Section 2 focuses on describing relationships between the models, however there are several examples of additional metadata that could be captured for each component model in the overall structural model:

- o origin and responsible party for maintenance of the model with contact information. In IETF standard models, the YANG 'organization' and 'contact' statement contents are a good example, but this is not necessarily the case for models from other sources.

- o license under which the model is distributed, e.g., open source or as part of a commercial license

- o classification of the model, including its category / subcategory, whether the model is intended to be used standalone, etc.

- o model dependencies, e.g., a list of other modules that are required

- o namespace information, including base namespace, prefixes, etc. to enable importing the model

- o pointer to the YANG code, if it is freely downloadable

- o implementation information, for example, a list of available implementations that support the model from vendors, open source projects, etc.

- o authentication information to allow users to verify that the model they download does in fact originate from the stated organization

For such an approach to be useful, we also require a registration system where model developers can register information about their models, and update it as needed. The IANA XML Registry" [RFC3688] provides a basic registry for YANG models, but the information is somewhat limited and is currently targeted at IETF-standardized models only. Further details on the proposal for such a registry may be forthcoming in further revisions to this document.

## 4.2. Service-layer composition

The proposed structural model covers a wide variety of components and protocols, and clearly not all of them are needed for all services. Another envisioned use case for the structural model is the ability to reference the set of models that are needed for specific use cases or services. The intent is that the set would be based on best operational practices as defined by users or operators who run such services.

One approach for this would be to define a ’service overlay’ model, for example for Layer 3 VPN services, that defines the set of required configuration and state models, such as VRFs, interfaces, BGP, policy, ACLs, and QoS. Similar overlay models can be defined for other services or use cases, for example, basic Internet operations such as adding new peers or customers, or setting up Layer 2 VPNs. Note these overlay models may be complementary to actual configuration models for such services, which may focus on providing an abstracted set of configuration or operational state variables, which would then be mapped onto device level variables. We leave discussion of such mapping mechanisms to future revisions.

# 5. Security Considerations

The model structure described in this document does not define actual configuration and state data, hence it is not directly responsible for security risks.

However, each of the component models that provide the corresponding configuration and state data should be considered sensitive from a security standpoint since they generally manipulate aspects of network configurations. Each component model should be carefully evaluated to determine its security risks, along with mitigations to reduce such risks.

## 6. IANA Considerations

This YANG model currently uses a temporary ad-hoc namespace. If it is placed or redirected for the standards track, an appropriate namespace URI will be registered in the IETF XML Registry" [RFC3688]. The YANG structure modules will be registered in the "YANG Module Names" registry [RFC6020].

## 7. YANG module

The model structure is described by the YANG module below.

### 7.1. Model structure

```
<CODE BEGINS> file model-structure.yang
module model-structure {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/structure";

  prefix "struct";

  // import some basic types


  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.com";

  description
    "This module describes a model structure for YANG
    configuration and operational state data models. Its intent is to
    describe how individual device protocol and feature models fit
    together and interact.";

  revision "2015-03-06" {
```

```
       description
         "Initial revision";
       reference "TBD";
     }

     // extension statements

     // feature statements

     // identity statements

     // typedef statements

     // grouping statements

     grouping info {
       description
         "base system information";

       container info {
         description
           "This container is for base system information, including
           device type (e.g., physcal or virtual), model, serial no.,
           location, etc.";

         leaf device-type {
           //TODO: consider changing to an identity if finer grained
           // device type classification is envisioned
           type enumeration {
             enum PHYSICAL {
               description "physical or hardware device";
             }
             enum VIRTUAL {
               description "virtual or software device";
             }
           }
           description
             "Type of the device, e.g., physical or virtual.  This node
             may be used to activate other containers in the model";
         }

       }
     }

     grouping hardware {
       description
         "hardware / vendor -specific data relevant to the platform";
```

```
      container hardware {
        description
          "This container is an anchor point for platform-specific
          configuration and operational state data.  It may be further
          organized into chassis, linecards, ports, etc.  It is
          expected that vendor or platform-specific augmentations
          would be used to populate this part of the device model";
      }
    }

    grouping l2-protocol-members {
      description "containers for each layer 2 protocol model";

      container vsi {
        description "virtual switch instance (or virtual forwarding
        instance) for use in PWE3 / VPLS services";

      }

      container ipv6-ndp {
        description "IPv6 neighbor discovery";
        reference "RFC 4861 - Neighbor Discovery for IP version 6
        (IPv6)";
      }

      container arp {
        description "Address resolution protocol";
        reference "STD 37 - An Ethernet Address Resolution Protocol";
      }

      container rstp {
        description "rapid spanning tree protocol";
        reference "IEEE 802.1D-2004";
      }

      container lldp {
        description "link layer discovery protocol";
        reference "IEEE 802.1AB";
      }

      container ptp {
        description
          "precision time protocol for time synchronization services.
          PTP also typically requires per-interface configuration";
        reference "IEEE 1588-2008";
      }
    }
```

```
     grouping l2-protocols {
       description "Layer 2 protocol models";

       container layer-2-protocols {
         description "layer 2 protocols and features";

         uses l2-protocol-members;

       }

     }

     grouping igp-protocol-members {
       description "containers for IGPs";

       container is-is {
         description "IS-IS IGP routing protocol";
         reference "RFC 1195 - Use of OSI IS-IS for Routing in TCP/IP
         and Dual Environments";
       }

       container ospf {
         description "OSPF IGP routing protocols";

         container ospf2 {
           description "OSPF v2";
           reference "RFC 2328 - OSPF Version 2";
         }

         container ospf3 {
           description "OSPF v3";
           reference "RFC 5340 - OSPF for IPv6";
         }
       }

       container igp-common {
         description "Common parameters for IGP protocols";
       }
     }

     grouping l3-protocol-members-vrf {
       description "containers for layer 3 protocol that are supported
       in a VRF instance";

       container bgp {
         description "BGP 4";
         reference "RFC 4271 - A Border Gateway Protocol 4 (BGP-4)";
       }
```

```
        container igp {
          description "interior gateway protocols";

          uses igp-protocol-members;
        }

        container bfd {
          description "bidirectional forwarding detection";
          reference "RFC 5880 - Bidirectional Forwarding Detection
          (BFD)";
        }

        container pim {
          description "protocol independent multicast";
          reference "RFC 4601 - Protocol Independent Multicast -
          Sparse Mode (PIM-SM): Protocol Specification (Revised)";
        }

        container igmp {
          description "Internet group management protocol";
          reference "RFC 3376 - Internet Group Management Protocol,
          Version 3";
        }

        container static-routes {
          description "static route that are manually created";
        }

      }

      grouping l3-protocols-misc {
        description "containers for other features operating at the
        network layer";

      }

      grouping l3-protocols-mpls {
        description "models related to MPLS and TE";

        container mpls-te {
          description "MPLS and traffic engineering";

          container global {
            description "global MPLS configuration";
          }

          container signaling {
            description "MPLS signaling protocols";
```

```
          container rsvp {
            description "RSVP signaling";
            reference "RFC 3209 - RSVP-TE: Extensions to RSVP for LSP
            Tunnels";
          }

          container segment-routing {
            description "SR signaling";
            reference "Segment Routing Architecture -
            draft-filsfils-spring-segment-routing-04";
          }

          container ldp {
            description "label distribution protocol";
            reference "RFC 5036 - LDP Specification";
          }
        }

        container label-switched-paths {
          description "models for different types of LSPs";

          container constrained-path {
            description "traffic-engineered, or constrained path LSPs";
          }

          container igp-congruent {
            description "LSPs that follow the IGP-computed path";
          }

          container static {
            description "statically configured LSPs";
          }
        }
      }
    }

    grouping l3-protocol-members {
      description "containers for all layer 3 protocols";

      uses l3-protocol-members-vrf;
      uses l3-protocols-misc;
      uses l3-protocols-mpls;

    }

    grouping l3-routing-policy {
      description "containers for routing policy models";
```

```
      container common {
        description "generic routing policy framework and
        configuration parameters";
      }

      container bgp-policy {
        description "BGP-specific routing policy parameters";
      }

      container igp-policy {
        description "IGP routing policy knobs -- may include
        policy parameters for specific IGPs";
      }

      container vrf-policy {
        description "import/export policies for VRFs";
      }
    }

    grouping l3-protocols {
      description "Layer 3 protocol models";

      container layer-3-protocols {
        description "layer 3 protocols and features";

        container global {
          description "router-wide instance of each routing protocol";

          uses l3-protocol-members;

        }

        list vrf {
          key vrf-name;
          description "list of VRF instances";

          leaf vrf-name {
            type string;
            description "name or id of the routing instance / VRF";
          }

          uses l3-protocol-members-vrf;
        }

        container routing-policy {
          description "models related to routing policy across
          protocols and VRFs";
```

```
          uses l3-routing-policy;

        }
      }
    }

    grouping interface-ip-common {
      description
        "interface-specific configuration for IP interfaces, IPv4 and
        IPv6";

      container vrrp {
        description "virtual router redundancy protocol";
        reference "RFC 5798 - Virtual Router Redundancy Protocol
        (VRRP) Version 3 for IPv4 and IPv6";
      }
    }

    grouping interface-addr-families {
      description
        "containers for addr family-specific data attached
        to interfaces";

      container ipv4 {
        description "IPv4 interfaces";

        uses interface-ip-common;
      }

      container ipv6 {
        description "IPv6 interfaces";

        uses interface-ip-common;
      }
    }

    grouping interfaces {
      description "interface-related models";

      container interfaces {
        description "various interface models";

        container ethernet {
          description "Ethernet interface config, e.g., 10, 40,
          100GBE";

          container aggregates {
            description "LAGs, LACP, etc. for Ethernet interfaces";
```

```
            reference "IEEE 802.1ad, 802.1AX";
          }

          container vlans {
            description "VLANs, 802.1q, q-in-q, etc.";
            reference "IEEE 802.1Q";
          }

          container lfm {
            description
              "Link-layer fault management for Ethernet interfaces";
            reference "IEEE 802.3ah";
          }
        }

        container sonet-sdh {
          description "SONET/SDH interfaces";
          reference
            "SDH: ITU standards G.707, G.783, G.784, and G.803
            SONET: ANSI standard T1.105";

        }

        container addressing {
          description "addressing and other interface-specific data,
          e.g., data plane protocols";

          uses interface-addr-families;
        }

        container tunnels {
          description
            "logical tunnel interfaces incl. GRE, VxLAN, L2TP etc.";
        }

      }
    }

    grouping oam {
      description "containers for features related to operations,
      administration, and management";

      container oam {
        description "commonly use OAM functions on devices";

        container snmp {
          description "SNMP server information, e.g., allowed clients";
        }
```

```
        container cfm {
          description
            "Ethernet connectivity fault management.  Also includes
            options that are associated with specific interfaces, such
            as maintenance endpoint domains.";
          reference "IEEE 802.1ag";
        }

        container twamp {
          description
            "Two-way active measurement protocol for measuring
            round-trip IP layer performance.";
          reference "RFC 5357 A Two-Way Active Measurement Protocol
          (TWAMP)";
        }
      }
    }

    grouping system-services {
      description "containers for system service models";

      container dns {
        description "domain name service and resolver configurration";
      }

      container ntp {
        description "network time protocol configuration";
      }

      container dhcp {
        description "dhcp and relay services";
      }

      container syslog {
        description "syslog configuration";
      }

      container ssh {
        description "ssh server configuration";
      }

      container stat-coll {
        description
          "mechanisms for data collection from devices, including
          packet and flow-level sampling";
      }

      uses oam;
```

```
     }

     grouping system-aaa {
       description "AAA-related services";

       container aaa {
         description "authentication, authorization, and accounting";

         container tacacs {
           description "TACACS+ configuration";
         }

         container radius {
           description "RADIUS";
           reference "RFC 2865 - Remote Authentication Dial In User
           Service (RADIUS)";
         }
       }
     }

     grouping system {
       description "system-wide services";

       container system {
         description "system services";

         uses system-services;
         uses system-aaa;

         container users {
           description "local user configuration";
         }
       }
     }

     grouping acl {
       description "forwarding rules";

       container acl {
         description "ACLs and packet forwarding rules";
       }
     }

     grouping qos {
       description "QoS features";

       container qos {
         description "QoS, including policing, shaping, etc.";
```

```
      }
    }

    // data definition statements

    container device {
      description "top-level anchor point for models.  Device is a
      generic L2/L3 network element";

      uses info;
      uses hardware;
      uses system;
      uses interfaces;
      uses acl;
      uses qos;

      container logical-routers {
        description "devices may support multiple logical router
        instances";

        list logical-router {

          key router-id;
          description "list of logical router instances";

          leaf router-id {
            type uint8; // expect a small number of logical routers
            description "identifier of the logical router instance";
          }

          leaf router-name {
            type string; // expect a small number of logical routers
            description "identifier of the logical router instance";
          }

          uses l2-protocols;
          uses l3-protocols;

        }
      }

    }

    // augment statements

    // rpc statements

    // notification statements
```

```
}
<CODE ENDS>
```

# 8. References

## 8.1. Normative references

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2014.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, June 2014.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

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

## 8.2. Informative references

[RTG-CFG] Lhotka, L., "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-16 (work in progress), October 2014.

[RTG-POLICY] Shaikh, A., Shakir, R., D’Souza, K., and C. Chase, "Routing Policy Configuration Model for Service Provider Networks", draft-shaikh-rtgwg-policy-model-00 (work in progress), January 2015.

[RTG-OSPF] Yeung, D., Qu, Y., Zhang, J., and D. Bogdanovic, "Yang Data Model for OSPF Protocol", draft-yeung-netmod-ospf-02 (work in progress), October 2014.

[RTG-ISIS] Litkowski, S., Yeung, D., Lindem, A., Zhang, J., and L. Lhotka, "YANG Data Model for ISIS protocol", draft-ietf-isis-yang-isis-cfg-01 (work in progress), October 2014.

## Appendix A. Acknowledgements

The authors are grateful for valuable contributions to this document and the associated models from: Deepak Bansal, Paul Borman, Chris Chase, Josh George, Marcus Hines, and Jim Uttaro.

## Authors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Rob Shakir
BT
pp. C3L, BT Centre
81, Newgate Street
London EC1A 7AJ
UK

Email: rob.shakir@bt.com
URI: http://www.bt.com/

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com

Luyuan Fang
Microsoft
205 108th Ave. NE, Suite 400
Bellevue, WA
US

Email: lufang@microsoft.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

R. Shakir
BT
A. Shaikh
M. Hines
Google

March 9, 2015

# Consistent Modeling of Operational State Data in YANG
draft-openconfig-netmod-opstate-00

## Abstract

This document proposes an approach for modeling configuration and operational state data in YANG that is geared toward network management systems that require capabilities beyond those typically envisioned in a NETCONF-based management system. The document presents the requirements of such systems and proposes a modeling approach to meet these requirements, along with implications and design patterns for modeling operational state in YANG.

## Status of This Memo



## Copyright Notice

## 1. Introduction

Retrieving the operational state of a network element (NE) is a critical process for a network operator, both because it determines how the network is currently running (for example, how many errors are occurring on a certain link, what is the load of that link); but also because it determines whether the intended configuration applied by a network management system is currently operational. Whilst changing of the configuration of NE may be a process which occurs relatively infrequently, the accessing of the state of the network is significantly more often - with knowledge of the real-time state of the network by external analysis and diagnostic systems being desired (implying reading of this data on the order of millisecond timescales).

It is desirable to model configuration and operational state within a single schema. This allows a network operator or management system to retrieve information as to the intended state of the network system (the configuration), and easily relate to this to the actual running state. There are numerous reasons that the intended state may not be reflected by the running config:

- o Protocol negotiations may be required for multiple NEs to agree on a certain running value - for example, the HOLDTIME of a BGP session is chosen by taking the minimum value of the locally configured and advertised value received from the remote speaker. The operational value of the HOLDTIME may therefore be lower than the configured value on the local system.

- o Where the application of a change is asynchronous - due to system operations, or a pre-requisite for another event to occur before the configuration value is applied (e.g., a protocol session restart) - then the intended configuration value may not determine whether the configuration has been committed to the running configuration; or whether the pre-requisite event has occurred.

Based on such differences between intended and running state, the operation of checking one state and then subsequently applying a change is very common. For example, checking the current IGP metric of a certain link, and if it is not reflective of the desired value, subsequently applying a change. Rather than viewing configuration and operational state separately, having both types of values in a common location within the same data schema is advantageous. In this

way, no complex mapping between the path where the value is read, and the path by which it is configured is required.

The majority of existing designs of the layout and presentation of a YANG [RFC6020] model considers only the semantics of the NETCONF protocol - however, it is advantageous that any data model (expressed in YANG) should be suitable for use with multiple protocols. Such protocols may be alternatives to NETCONF - e.g., RESTCONF - but also may be specifically engineered for accessing particular operational data (e.g., streamed data from a network element, rather than 'SNMP-like' polled mechanisms).

Based on the inherent link between the configuration and state data for a NE, and the importance of state for a network operator, YANG's focus solely on configuration data is suboptimal[RFC6244]. We therefore consider that there is a requirement to consider (and define common approaches for) the definition of state and operational data within a YANG model. Such considerations should be cognisant of the protocols used to interact with the data schema.

## 2. Operational requirements

Our proposal is motivated by a number of operational requirements as described below.

### 2.1. Intended configuration as part of operational state

The definition of operational state in [RFC6244] includes read-only transient data that is the result of system operation or protocol interactions, and data that is typically thought of as counters or statistics. In many operational use cases it is also important to distinguish between the intended value of a configuration variable and its actual configured state, analogous to candidate and running configuration, respectively, in NETCONF datastores. In non-transactional or asynchronous environments, for example, these may be different and it is important to know when they are different or when they have converged (see requirement #2). For this reason, we consider the intended configuration as an additional important element of the operational state. This is not considered in [RFC6244].

### 2.2. Support for both transactional, synchronous management systems as well as distributed, asynchronous management systems

In a synchronous system, configuration changes are transactional and committed as an atomic unit. This implies that the management system knows the success or failure of the configuration change based on the return value, and hence knows that the intended configuration matches

what is on the system. In particular, the value of any configuration variable should always reflect the (intended) configured value. Synchronous operation is generally associated with a NETCONF-based system that provides transactional semantics for all changes.

In an asynchronous system, configuration changes to the system may not be reflected immediately, even though the change operation returns success. Rather, the change is verified by observing the state of the system, for example based on notifications, or continuously streamed values of the state. In this case, the value of a configuration variable may not reflect the intended configured value at a given point in time.

The asynchronous use case is important because synchronous operation may not always be possible. For example, in a large scale environment, the management system may not need to wait for all changes to complete if it is acceptable to proceed while some configuration values are being updated. In addition, not all devices may support transactional changes, making asynchronous operation a requirement. Moreover, using observed state to infer the configured value allows the management system to learn the time taken to complete various configuration changes.

### 2.3. Separation of configuration and operational state data; ability to retrieve them independently

These requirements are also mentioned in [RFC3535]:

- It is necessary to make a clear distinction between configuration data, data that describes operational state and statistics.

- It is required to be able to fetch separately configuration data, operational state data, and statistics from devices, and to be able to compare these between devices.

### 2.4. Ability to retrieve operational state corresponding only to derived values, statistics, etc.

When the management system operates in synchronous mode, it should be able to retrieve only the operational state corresponding to the system determined values, such as negotiated values, protocol determined values, or statistics and counters. Since in synchronous mode the intended and actual configuration values are identical, sending the intended configuration state is redundant.

### 2.5. Consistent schema locations for configuration and corresponding operational state data

This requirement implies that a common convention is used throughout the schema to locate configuration and state data so that the management system can infer how to access one or the other without needing significant external context. When considering intended configuration as part of operational state (as discussed in Section 2.1), it is similarly required that the intended value vs. actual value for a particular configuration variable should be possible to locate with minimal, if any, mapping information.

This requirement becomes more evident when considering the composition of individual data models into a higher-level model for a complete device (e.g., /device[name=devXY]/protocols/routing/...) or even higher layer models maintained by network operators (e.g., /operatorX/global/continent[name=eur]/pop[name=paris]/device[name=devXY]/...). If each model has it's own way to separate configuration and state data, then this information must be known at potentially every subtree of the composed model.

## 3. Implications on modeling operational state

The requirements in Section 2 give rise to a number of new considerations for modeling operational state. Some of the key implications are summarized below.

### 3.1. Inclusion of intended configuration as part of operational state

This implies that a copy of the configurable (i.e., writable) values should be included as read-only variables in containers for operational state, in addition to the variables that are traditionally thought of as state variables (counters, negotiated values, etc.).

### 3.2. Corresponding leaves for configuration and state

Any configuration leaf should have a corresponding state leaf. The opposite is clearly not true -- some parts of the model may only have derived state variables, for example the contents of a routing table that is populated by a dynamic routing protocols like BGP or IS-IS.

### 3.3. Retrieval of only the derived, or NE-generated part of the operational state

YANG and NETCONF do not currently differentiate between state that is derived by the NE, state representing statistics, and state representing intended configuration -- all state is simply marked as

'config false' or read-only. To retrieve only the state that is not part of intended configuration, we require a new way to tag such data. This is proposed in this document as a YANG extension. Alternatively, as described in [RFC6244], a new NETCONF datastore for operational state that is just for NE- generated state could also be used to allow <get> (or similar) operations to specify just that part of the state.

### 3.4. Consistency and predictability in the paths where corresponding state and configuration data may be retrieved

To avoid arbitrary placement of state and configuration data containers, the most consistent options would be at the root of the model (as done in [YANG-IF]) or at the leaves, i.e., at the start or end of the paths. When operators compose models into a higher level model, the root of the model is no longer well-defined, and hence neither is the start of the path. For these reasons, we propose placing configuration and state separation at leaves of the model.

### 3.5. Reuse of existing NETCONF conventions where applicable

Though not a specific requirement, models for operational state should take advantage of existing protocol mechanisms where possible, e.g., to retrieve configuration and state data.

## 4. Proposed operational state structure

Below we show an example model structure that meets the requirements described above for all four types of data we are considering:

- o configuration (writable) data
- o operational state data on the NE that is derived, negotiated, set by a protocol, etc.
- o operational state data for counters or statistics
- o operational state data representing intended configuration

### 4.1. Example model structure

The example below shows a partial model (in ascii tree format) for managing Ethernet aggregate interfaces (leveraging data definitions from [RFC7223]):

```
+--rw interfaces
    +--rw interface* [name]
       +--rw name       -> ../config/name
       +--rw config
       |    ...
       +--ro state
       |  | ...
       |  +--ro counters
       |     +--ro discontinuity-time    yang:date-and-time
       |     +--ro in-octets?            yang:counter64
       |     +--ro in-unicast-pkts?      yang:counter64
       |     +--ro in-broadcast-pkts?    yang:counter64
       |     +--ro in-multicast-pkts?    yang:counter64
       |     +--ro in-discards?          yang:counter64
       |     +--ro in-errors?            yang:counter64
       |     +--ro in-unknown-protos?    yang:counter64
       |     +--ro out-octets?           yang:counter64
       |     +--ro out-unicast-pkts?     yang:counter64
       |     +--ro out-broadcast-pkts?   yang:counter64
       |     +--ro out-multicast-pkts?   yang:counter64
       |     +--ro out-discards?         yang:counter64
       |     +--ro out-errors?           yang:counter64
       +--rw aggregation!
          +--rw config
          |  +--rw lag-type?    aggregation-type
          |  +--rw min-links?   uint16
          +--ro state
          |  +--ro lag-type?    aggregation-type
          |  +--ro min-links?   uint16
          |  +--ro members*     ocif:interface-ref
          +--rw lacp!
             +--rw config
             |  +--rw interval?   lacp-period-type
             +--rw members* [interface]
             |  +--rw interface    ocif:interface-ref
             |  +--ro state
             |     +--ro activity?          lacp-activity-type
             |     +--ro timeout?           lacp-timeout-type
             |     +--ro synchronization?   lacp-synch-type
             |     +--ro aggregatable?      boolean
             |     +--ro collecting?        boolean
             |     +--ro distributing?      boolean
             +--ro state
                +--ro interval?   lacp-period-type
```

In this model, the path to the configurable (rw) items at the aggregate interface level is:

/interfaces/interface[name=ifName]/aggregation/config/...

The corresponding operational state is located at:

/interfaces/interface[name=ifName]/aggregation/state/...

This container holds a read-only copy of the intended configuration variables (lag-type and min-links), as well as a generated list of member interfaces (the members leaf-list) for the aggregate that is active when the lag-type indicates a statically configured aggregate. Note that although the paths to config and state containers are symmetric, the state container contains additional derived variables.

The model has an additional hierarchy level for aggregate interfaces that are maintained using LACP. For these, the configuration path is:

/interfaces/interface[name=ifName]/aggregation/lacp/config/...

with the corresponding state container (in this case with only the state corresponding to the intended configuration) at:

/interfaces/interface[name=ifName]/aggregation/lacp/state/...

There is an additional list of members for LACP-managed aggregates with only a state container:

/interfaces/interface[name=ifName]/aggregation/lacp/
members[name=ifName]/state/...

Note that it is not required that both a state and a config container be present at every leaf. It may be convenient to include an empty config container to make it more explicit to the management system that there are no configuration variables at this location in the data tree.

Finally, we can see that the generic interface object also has config and state containers (these are abbreviated for clarity). The state container has a subcontainer for operational state corresponding to counters and statistics that are valid for any interface type:

/interfaces/interface[name=ifName]/state/counters/...

## 5. Impact on model authoring

One drawback of structuring operational and configuration data in this way is the added complexity in authoring the models, relative to the way some models are currently built with state and config split

at the root of the individual model (e.g., in [RFC7223], [RFC7317], and [IETF-RTG]). Moving the config and state containers to each leaf adds a one-time modeling effort, which is somewhat dependent on the model structure itself (how many layers of container hierarchy, number of lists, etc.) However, we feel this effort is justified by the resulting simplicity with which management systems can access and correlate state and configuration data.

### 5.1. Modeling design patterns

We propose some specific YANG modeling design patterns that may be useful for building models following these conventions.

#### 5.1.1. Basic structure

Since leaves that are created under the 'config' container are duplicated under the 'state' container, it is recommended that the following conventions are used to ensure that the schema remain as simple as possible:

- o A grouping for the 'config' data items is created - with a specific naming convention to indicate that such variables are configurable, such as a suffix like '-config' or '_config'. For example, the OpenConfig BGP model [OC-BGP] adopts the convention of appending "_config" to the name of the container.

- o A grouping for the 'state' data items is created, with a similar naming convention as above, i.e., with a suffix such as '-state' or '_state'. The BGP model uses "_state".

- o A structure grouping is created that instantiates both the 'config' and 'state' containers. The 'config' container should include the "-config" grouping, whilst the state container has both the "-config" and "-state" groupings, along with the 'config false' statement.

A simple example in YANG is shown in Appendix B.

#### 5.1.2. Handling lists

In YANG 1.0, lists have requirements that complicate the creation of the parallel configuration and state data structures. First, keys must be children of the list; they cannot be further down the data hierarchy within a subsequent container. For example, the 'interface' list cannot be keyed by /interfaces/interface/config/name. Second YANG requires that the list key is part of the configuration or state data in each list member.

We consider two possible approaches for lists:

1. list keys appear only at the top level of the list, i.e., not duplicated under the 'config' or 'state' containers within the list

2. the data represented by the list key appears in the config and state containers, and a key with type leafref is used in the top level of the list pointing to the corresponding data node in the config (or state) container.

Option 1 has the advantage of not duplicating data, but treats the data item (or items) that are keys as special cases, i.e., not included in the config or state containers. Option 2 is appealing in that configurable data always appears in the config container, but requires an arguably unnecessary key pointing to the data from the top level of the list.

Appendix C shows a simple example of both options.

#### 5.1.3. Selective use of state data from common groupings

In a number of cases, it is desirable that the same grouping be used within different places in a model - but state information is only relevant in one of these paths. For example, considering BGP, peer configuration is relevant to both a "neighbor" (i.e., an individual BGP peer), and also to a peer-group (a set of peers). Counters relating to the number of received prefixes, or queued messages, are relevant only within the 'state' container of the peer (rather than the peer-group). In this case, use of the 'augment' statement to add specific leaves to only one area of the tree is recommended, since it allows a common container to be utilized otherwise.

#### 5.1.4. Non-corresponding configuration and state data

There are some instances where only an operational state container is relevant without a corresponding configuration data container. For example, the list of currently active member interfaces in a LACP-managed LAG is typically reported by the system as operational state that is governed by the LACP protocol. Such data is not directly configured. Similarly, counters and statistics do not have corresponding configuration. In these cases, we can either omit the config container from such leaves, or provide an empty container as described earlier. With both options, the management system is able to infer that such data is not configurable.

## 6. YANG language considerations

In adopting the approach described in this document for modeling operational state data in YANG, we encounter several language limitations that are described below. We disucss some initial thoughts on possible changes to the language to more easily enable the proposed model for operational state modeling.

### 6.1. Distinguishing derived operational state data and intended configuration

As mentioned in Section 2, we require a way to separately query operational state that is not part of intended configuration (e.g., protocol-determined data, counters, etc.). YANG and NETCONF do not distinguish types of operational state data, however. To overcome this, we currently use a YANG language extension to mark such data as ’operational: true’. Ideally, this could be generalized beyond the current ’config: true / false’ to something like "operational: false", "operational: intent", and "operational:true".

### 6.2. YANG lists as maps

YANG has two list constructs, the ’leaf-list’ which is similar to a list of scalars in other programming languages, and the ’list’ which allows a keyed list of complex structures, where the key is also part of the data values. As described in Section [impact], the current requirements on YANG list keys require either duplication of data, or treating some data (i.e., those that comprise list keys) as a special case. One solution is to generalize lists to be more like map data structures, where each list member has a key that is not required to part of the configuration or state data. This allows list keys to be arbitrarily defined by the user if desired, or based on values of data nodes. In the latter case, the specification of which data nodes are used in constructing the list key could be indicated in the meta-data associated with the key.

### 6.3. Configuration and state data hierarchy

YANG does not allow read-write configuration data to be child nodes of read-only operational state data. This requires the definition of separate state and config containers as described above. However, it may be desirable to simplify the schema by ’flattening’, e.g., having the operational state as the root of the data tree, with only config containers needed to specify the variables that are writable (in general, the configuration data is much smaller than operational state data). Naming the containers explicitly according the config / state convention makes the intent of the data clear, and should allow relaxing of the current YANG restrictions. That is, a read-write

config container makes explicit the nature of the enclosed data even if the parent data nodes are read-only. This of course requires that all data in a config container are in fact configurable -- this is one of the motivations of pushing such containers as far down in the schema hierarchy as possible.

## 7. Security Considerations

This document addresses the structure of configuration and operational state data, both of which should be considered sensitive from a security standpoint. Any data models that follows the proposed structuring must be carefully carefully evaluated to determine its security risks. In general, access to both configuration (write) and operational state (read) data must be carefully controlled through appropriate access control and authorization mechanisms.

## 8. References

### 8.1. Normative references

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

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, June 2011.

[RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

### 8.2. Informative references

[IETF-RTG] Lhotka, L., "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-16 (work in progress), October 2014.

[OC-BGP] Shaikh, A., D’Souza, K., Bansal, D., and R. Shakir, "BGP Configuration Model for Service Provider Networks", draft-shaikh-idr-bgp-model-01 (work in progress), March 2015.

## Appendix A. Acknowledgements

The authors are grateful for valuable input to this document from: Martin Bjorklund, Paul Borman, Chris Chase, Feihong Chen, Josh George, Carl Moberg, Jason Sterne, and Jim Uttaro.

## Appendix B. Example YANG base structure

Below we show an example of the basic YANG building block for organizing configuration and operational state data as described in Section 4

```
grouping example-config {
    description "configuration data for example container";

    leaf conf-1 {
      type empty;
    }

    leaf conf-2 {
      type string;
    }
  }

  grouping example-state {
    description
      "operational state data (derived, counters, etc.) for example
      container";

    leaf state-1 {
      type boolean;
    }

    leaf state-2 {
      type string;
    }

    container counters {
      description
        "operational state counters for example container";

      leaf counter-1 {
        type uint32;
      }

      leaf counter-2 {
        type uint64;
      }
```

```
      }
    }

    grouping example-structure {
      description
        "top level grouping for the example container -- this is used
        to put the config and state subtrees in the appropriate
        location";

      container example {
        description
          "top-level container for the example data";

        container config {

          uses example-config;

        }

        container state {

          config false;
          uses example-config;
          uses example-state;
        }
      }
    }

    uses example-structure;
```

The corresponding YANG data tree is:

```
     +--rw example
        +--rw config
        |  +--rw conf-1?   empty
        |  +--rw conf-2?   string
        +--ro state
           +--ro conf-1?     empty
           +--ro conf-2?     string
           +--ro state-1?    boolean
           +--ro state-2?    string
           +--ro counters
              +--ro counter-1?   uint32
              +--ro counter-2?   uint64
```

## Appendix C. Example YANG list structure

As described in Section 5.1.2, there are two options we consider for building lists according to the proposed structure. Both are shown in the example YANG snippet below. The groupings defined above in Appendix B are reused here.

```
grouping example-no-conf2-config {
    description
    "configuration data for example container but without the conf-2
    data leaf which is used as a list key";

    leaf conf-1 {
      type empty;
    }

  }

  grouping example-structure {
    description
      "top level grouping for the example container -- this is used
      to put the config and state subtrees in the appropriate
      location";

    list example {

      key conf-2;
      description
        "top-level list for the example data";

      leaf conf-2 {
        type leafref {
          path "../config/conf-2";
        }
      }

      container config {

        uses example-config;

      }

      container state {

        config false;
        uses example-config;
        uses example-state;
      }
```

```
      }

      list example2 {

        key conf-2;
        description
          "top-level list for the example data";

        leaf conf-2 {
          type string;
        }

        container config {

          uses example-no-conf2-config;

        }

        container state {

          config false;
          uses example-no-conf2-config;
          uses example-state;
        }
      }
    }

    uses example-structure;
```

The corresponding YANG data tree is shown below for both styles of lists.

```
   +--rw example* [conf-2]
      |  +--rw conf-2    -> ../config/conf-2
      |  +--rw config
      |  |  +--rw conf-1?   empty
      |  |  +--rw conf-2?   string
      |  +--ro state
      |     +--ro conf-1?     empty
      |     +--ro conf-2?     string
      |     +--ro state-1?    boolean
      |     +--ro state-2?    string
      |     +--ro counters
      |        +--ro counter-1?   uint32
      |        +--ro counter-2?   uint64
      +--rw example2* [conf-2]
         +--rw conf-2    string
         +--rw config
         |  +--rw conf-1?   empty
         +--ro state
            +--ro conf-1?     empty
            +--ro state-1?    boolean
            +--ro state-2?    string
            +--ro counters
               +--ro counter-1?   uint32
               +--ro counter-2?   uint64
```

## Authors' Addresses

Rob Shakir
BT
pp. C3L, BT Centre
81, Newgate Street
London  EC1A 7AJ
UK

Email: rob.shakir@bt.com
URI:   http://www.bt.com/

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA  94043
US

Email: aashaikh@google.com

Marcus Hines
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: hines@google.com

NETCONF Data Modeling Language Working Group (netmod)
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

E. Voit
A. Clemm
Cisco Systems
S. Mertens
Prismtech

March 9, 2015

# Requirements for Peer Mounting of YANG subtrees from Remote Datastores
# draft-voit-netmod-peer-mount-requirements-02

## Abstract

Network integrated applications want simple ways to access YANG objects and subtrees which might be distributed across network. Performance requirements may dictate that it is unaffordable for a subset of these applications to go through existing centralized management brokers. For such applications, development complexity must be minimized. Specific aspects of complexity developers want to ignore include:

- o whether authoritative information is actually sourced from remote datastores (as well as how to get to those datastores),
- o whether such information has been locally cached or not,
- o whether there are zero, one, or more controllers asserting ownership of information, and
- o whether there are interactions with other applications concurrently running elsewhere

The solution requirements described in this document detail what is needed to support application access to authoritative network YANG objects from controllers (star) or peering network devices (mesh) in such a way to meet these goals.

## Status of This Memo

## Copyright Notice



## Table of Contents

## 1. Business Problem

Instrumenting Physical and Virtual Network Elements purely along device boundaries is insufficient for today's requirements. Instead, users, applications, and operators are asking for the ability to interact with varying subsets of network information at the highest viable level of abstraction. Likewise applications that run locally on devices may require access to data that transcends the boundaries of the device they are deployed. Achieving this can be difficult since a running network is comprised of a distributed mesh of object ownership. (I.e., the authoritative device owning a particular object will vary.) Solutions require the transparent assembly of different objects from across a network in order to provide consolidated, time synchronized, and consistent views required for that abstraction.

Recent approaches have focused on a Network Controller as the arbiter of new network-wide abstractions. Controller based solutions are supportable by requirements outlined in this document. However this is not the only deployment model covered by this document. Equally valid are deployment models where Network Elements exchange information in a way which allows one or more of those Elements to provide the desired network level abstraction. This is not a new idea. Examples of Network Element based protocols which already do network level abstractions include VRRP [RFC3768], mLACP/ICCP[ICCP], and Anycast-RP [RFC4610] . As network elements increase their compute power and support Linux based compute virtualization, we should expect additional local applications to emerge as well (such as Distributed Analytics [1]).

Ultimately network application programming must be simplified. To do this:

- o we must provide APIs to both controller and network element based applications in a way which allows access to network objects as if they were coming from a cloud,

- o we must enable these local applications to interact with network level abstractions,

- o we must hide the mesh of interdependencies and consistency enforcement mechanisms between devices which will underpin a particular abstraction,

- o we must enable flexible deployment models, in which applications are able to run not only on controller and OSS frameworks but also on network devices without requiring heavy middleware with large footprints, and

- o we need to maintain clear authoritative ownership of individual data items while not burdening applications with the need to reconcile and synchronize information replicated in different systems, nor needing to maintain redundant data models that operate on the same underlying data.

These steps will eliminate much unnecessary overhead currently required of today’s network programmer.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Authoritative Datastore - A datastore containing the authoritative copy of an object, i.e. the source and the "owner" of the object.

Client Datastore - a datastore containing an object whose source and "owner" is a remote datastore.

Data Node - An instance of management information in a YANG datastore.

Datastore - A conceptual store of instantiated information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Data Subtree - An instantiated data node and the data nodes that are hierarchically contained within it.

Mount Client - The system at which the mount point resides, into which on or more remote subtrees may be mounted.

Mount Binding - An instance of mounting from a specific Mount Point to a remote datastore. Types include:

- o On-demand: Mount Client only pulls information when application requests

- o Periodic: Mount Server pushes current state at a pre-defined interval

- o Unsolicited: Mount Server maintains active bindings and sends to client cache upon change

Mount Point - Point in the local data store which may reference a single remote subtree

Mount Server - The server with which the Mount Client communicates and which provides the Mount Client with access to the mounted information. Can be used synonymously with Mount Target.

Peer Mount - The act of representing remote objects in the local datastore

Target Data Node - Data Node on Mount Server against which a Mount Binding is established

## 3. Solution Context

YANG modeling has emerged as a preferred way to offer network abstractions. The requirements in this document can be enabled by expanding of the syntax of YANG capabilities embodied within RFC 6020 [RFC6020] and YANG 1.1 [rfc6020bis]. A companion draft to this one which details a potential set of YANG technology extensions which can support key requirements within this document are contained in . [draft-clemm-mount].

To date systems built upon YANG models have been missing two capabilities:

1. Peer Datastore Mount: Datastores have not been able to proxy objects located elsewhere. This puts additional burden upon applications which then need to find and access multiple (potentially remote) systems.

2. Eventual Consistency: YANG Datastore implementations have typically assumed ACID [2] transaction models. There is nothing

inherent in YANG itself which demands ACID transactional guarantees. YANG models can also expose information which might be in the process of undergoing convergence. Since IP networking has been designed with convergence in mind, this is a useful capability since some types of applications must participate where there is dynamically changing state.

### 3.1. Peer Mount

First this document will dive deeper into Peer Datastore Mount (a.k.a., "Peer Mount"). Contrary to existing YANG datastores, where hierarchical datatree(s) are local in scope and only includes data that is "owned" by the local system, we need an agent or interface on one system which is able refer to managed resources that reside on another system. This allows applications on the same system as the YANG datastore server, as well as remote clients that access the datastore through a management protocol such as NETCONF, to access all data as if it were local to that same server. This must be done in a manner that is transparent to users and applications. This must be done in a way which does not require a user or application to be aware of the fact that some data resides in a different location and have them directly access that other system. In this way, the user is projected an image of one virtual consolidated datastore.

The value in such a datastore comes from its under-the-covers federation. The datastore transparently exposes information from multiple systems across the network. The user does not need to be aware of the precise distribution and ownership of data themselves, nor is there a need for the application to discover those data sources, maintain separate associations with them, and partition its operations to fit along remote system boundaries. The effect is that a network device can broaden and customize the information available for local access. Life for the application is easier.

Any Object type can be included in such a datastore. This can include configuration data that is either persistent or ephemeral, and which is valid within only a single device or across a domain of devices. This can include operational data that represents state across a single device or across a multiple devices.

Another useful aspect of "Peer Mount" is its ability to embed information from external YANG models which haven’t necessarily been normalized. Normalization is a good thing. But the massive human efforts invested in uber-data-models have never gained industry traction due to the resulting models’ brittle nature and complexity. By mounting remote trees/objects into local datastores it is possible to expose remote objects under a locally optimized hierarchy without having to transpose remote objects into a separate local model. Once

this exists, object translation and normalization become optional capabilities which may also be hidden.

Another useful aspect of "Peer Mount" is its ability to mount remote trees where the local datastore does not know the full subtree being installed. In fact, the remote datastore might be dynamically changing the mounted tree. These dynamic changes can be reflected as needed under the "attachment points" within the namespace hierarchy where the data subtrees from remote systems have been mounted. In this case, the precise details of what these subtrees exactly contain does not need to be understood by the system implementing the attachment point, it simply acts as a single point of entry and "proxy" for the attached data.

### 3.2. Eventual Consistency and YANG 1.1

The CAP theorem [3] states that it is impossible for a distributed computer system to simultaneously provide Consistency, Availability, and Partition tolerance. (I.e., distributed network state management is hard.) Mostly for this reason YANG implementations have shied away from distributed datastore implementations where ACID transactional guarantees cannot be given. This of course limits the universe of applicability for YANG technology.

Leveraging YANG concepts, syntax, and models for objects which might be happening to undergo network convergence is valuable. Such reuse greatly expands the universe of information visible to networking applications. The good news is that there is nothing in YANG 1.1 syntax that prohibits its reapplication for distributed datastores. Extensions are needed however.

Requirements described within this document can be used to define technology extensions to YANG 1.1 for remote datastore mounting. Because of the CAP theorem, it must be recognized that systems built upon these extensions MAY choose to support eventual consistency rather than ACID guarantees. Some applications do not demand ACID guarantees (examples are contained in this document’s Use Case section). Therefore for certain classes of applications, eventual consistency [4] should be viewed as a cornerstone feature capability rather than a bug.

Other industries have been able to identify and realize the value in such model. The Object Management Group Data-Distribution Service for Real-Time Systems has even standardized these capabilities for non-YANG deployments [OMG-DDS]. Commercial deployments exist.

## 4. Example Use Cases

Many types of applications can benefit from the simple and quick availability of objects from peer network devices. Because network management and orchestration systems have been fulfilling a subset of the requirements for decades, it is important to focus on what has changed. Changes include:

- o SDN applications wish to interact with local datastore(s) as if they represent the real-time state of the distributed network.
- o Independent sets of applications and SDN controllers might care about the same authoritative data node or subtree.
- o Changes in the real-time state of objects can announce themselves to subscribing applications.
- o The union of an ever increasing number of abstractions provided from different layers of the network are assumed to be consistent with each other (at least once a reasonable convergence time has been factored in).
- o CPU and VM improvements makes running Linux based applications on network elements viable.

Such changes can enable a new class of applications. These applications are built upon fast-feedback-loops which dynamically tune the network based on iterative interactions upon a distributed datastore.

### 4.1. Cloud Policer

A Cloud Policer enables a single aggregated data rate to tenants/users of a data center cloud that applies across their VMs; a rate independent of where specific VMs are physically hosted. This works by having edge router based traffic counters available to a centralized application, which can then maintain an aggregate across those counters. Based on the sum of the counters across the set of edge routers, new values for each device based Policer can be recalculated and installed. Effectively policing rates are continuously rebalanced based on the most recent traffic offered to the aggregate set of edge devices.

The cloud policer provides a very simple cloud QoS model. Many other QoS models could also be implemented. Example extensions include:

- o CIR/PIR guarantees for a tenant,

- o hierarchical QoS treatment,

- o providing traffic delivery guarantees for specific enterprise branch offices, and

- o adjusting the prioritization of one application based on the activity of another application which perhaps is in a completely different location.

It is possible to implement such a cloud policer application with maximum application developer simplicity using peer mount. To do this the application accesses a local datastore which in turn does a peer mount from edge routers the objects which house current traffic counter statistics. These counters are accessed as if they were part of the local datastore structures, without concern for the fact that the actual authoritative copies reside on remote systems.

Beyond this centralized counter collection peer mount, it is also possible to have distributed edge routers mount information in the reverse direction. In this case local edge routers can peer mount centrally calculated policer rates for the device, and access these objects as if they were locally configured.

For both directions of mounting, the authoritative copy resides in a single system and is mounted by peers. Therefore issues with regards to inconsistent configuration of the same redundant data across the network are avoided. Also as can be seen in this use case, the same system can act as a mount client of some objects while acting as server for other objects.

### 4.2. DDoS Thresholding

Another extension of the "Cloud Policer" application is the creation of additional action thresholds at bandwidth rates far greater than might be expected. If these higher thresholds are hit, it is possible to connect in DDoS scrubbers to ingress traffic. This can be done in seconds after a bandwidth spike. This can also be done if non-bandwidth counters are available. For example, if TCP flag counts are available it is possible to look for changes in SYN/ACK ratios which might signal a different type of attack. In all cases, when network counters indicate a return to normal traffic profiles the DDoS Scrubbers can be automatically disconnected.

Benefits of only connecting a DDoS scrubber in the rare event an attack might be underway include:

- o marking down traffic for an out-of-profile tenant so that an potential attack doesn’t adversely impact others,

- o applying DDoS Scrubbing across many devices when an attack is detected in one,

- o reducing DDoS scrubber CPU, power, and licensing requirements (during the vast majority of time, spikes are not occurring), and

- o dynamic management and allocation of scarce platform resources (such as optimizing span port usage, or limiting IP-FIX reporting to levels where devices can do full flow detail exporting).

### 4.3. Service Chain Classification, Load Balancing and Capacity Management

Service Chains will dynamically change ingress classification filters, allocate paths from many ingress devices across shared resources. This information needs to be updated in real time as available capacity is allocated or failures are discovered. It is possible to simplify service chain configuration and dynamic topology maintenance by transparently updating remote cached topologies when an authoritative object is changed within a central repository. For example if the CPU in one VM spikes, you might want to recalculate and adjust many chained paths to relieve the pressure. Or perhaps after the recalculation you want to spin up a new VM, and then adjust chains when that capacity is on-line.

A key value here is central calculation and transparent auto-distribution. In other words, a change only need be updated by an application in a single location, and the infrastructure will automatically synchronize changes across any number of subscribing devices without application involvement. In fact, the application need not even know many devices are monitoring the object which has been changed.

Beyond 1:n policy distribution, applications can step back from aspects of failure recovery. What happens if a device is rebooting or simply misses a distribution of new information? With peer mount there is no doubt as to where the authoritative information resides if things get out of synch.

While this ability is certainly useful for dynamic service chain filtering classification and next hop mapping, this use case has more general applicability. With a distributed datastore, diverse applications and hosts can locally access a single device’s current VM CPU and Bandwidth values. They can do it without needing to explicitly query that remote machine. Updates from a device would come from a periodic push of stats to a transparent cache to subscribed, or via an unsolicited update which is only sent when these value exceed established norms.

## 5. Requirements

To achieve the objectives described above, the network needs to support a number of requirements

### 5.1. Application Simplification

A major obstacle to network programmability are any requirements which force applications to use abstractions more complicated than the developer cares to touch. To simplify applications development and reduce unnecessary code, the following needs must be met.

Applications MUST be able to access a local datastore which includes objects whose authoritative source is located in a remote datastore hosted on a different server.

Local datastores MUST be able to provide a hierarchical view of objects assembled from objects whose authoritative source may originate from potentially different and overlapping namespaces.

Applications MUST be able to access all objects of a datastore without concern where the actual object is located, i.e. whether the authoritative copy of the object is hosted on the same system as the local datastore or whether it is hosted in a remote datastore.

With two exceptions, a datastore’s application facing interfaces MUST make no differentiation whether individual objects exposed are authoritatively owned by the datastore or mounted from remote. This includes Netconf and Restconf as well as other, possibly proprietary interfaces (such as, CLI generated from corresponding YANG data models). The two exceptions are that it is acceptable to make a distinction between an object authoritatively owned by the data store and a remote object as follows:

- o Object updates / editing, creation and deletion. E.g. via edit-config conditions and constraints are assessed at the authoritative datastore when the update/create/delete is conducted. Any conditions or constraints at remote client datastores are NOT assessed.

- o Locks obtained at a client datastore: It is conceivable for the interface to distinguish between two lock modes: locking the entire subtree including remote data (in which case the datastore’s mount client needs to explicitly obtain and release locks from mounted authoritative datastores), or locking only authoritatively owned data, excluding remote data from the lock.

These exceptions should not be very problematic as non-authoritative copies will typically be marked as read-only. This will not violate any considerations of "no differentiation" of local or remote.

When a change is made to an object, that change will be reflected in any datastore in which the object is included. This means that a change made to the object through a remote datastore will affect the object in the authoritative datastore. Likewise, changes to an object in the authoritative datastore will be reflected at any client datastores.

The distributed datastore MUST be able to include objects from multiple remote datastores. The same object may be included in multiple remote datastores; in other words, an object’s authoritative datastore MUST support multiple clients.

The distributed datastore infrastructure MUST enable to access to some subset of the same objects on different devices. (This includes multiple controllers as well as multiple physical and virtual peer devices.)

Applications SHOULD be able to extract a time synchronized set of operational data from the datastore. (In other words, the application asks for a subset of network state at time-stamp or time-range "X". The datastore would then deliver time synchronized snapshots of the network state per the request. The datastore may work with NTP and operational counter to optimize the synchronization results of such a query. It is understood that some types of data might be undergoing convergence conditions.)

Authoritative datastore retain full ownership of "their" objects. This means that while remote datastores may access the data, any modifications to objects that are initiated at those remote datastores need to be authorized by the authoritative owner of the data. Likewise, the authoritative owner of the data may make changes to objects, including modifications, additions, and deletions, without needing to first ask for permission from remote clients.

Applications MUST be designed to deal with incomplete data if remote objects are not accessible, e.g. due to temporal connectivity issues preventing access to the authoritative source. (This will be true for many protocols and programming languages. Mount is unlikely to add anything new here unless applications have extra error handling routines to deal with when there is no response from a remote system.).

## 5.2. Caching

Remote objects in a datastore can be accessed "on demand", when the application interacting with the datastore demands it. In that case, a request made to the local datastore is forwarded to the remote system. The response from the remote system, e.g. the retrieved data, is subsequently merged and collated with the other data to return a consolidated response to the invoking application.

A downside of a datastore which is distributed across devices can be the latency induced when remote object acquisition is necessary. There are plenty of applications which have requirements which simply cannot be served when latency is introduced. The good news is that the concept of caching lends itself well to distributed datastores. It is possible to transparently store some types of objects locally even when the authoritative copy is remote. Instead of fetching data on demand when an application demands it, the application is simply provided with the local copy. It is then up to the datastore infrastructure to keep selected replicated info in synch, e.g. by prefetching information, or by having the remote system publish updates which are then locally stored. At this point, it is expected that a preferred method of subscribing to and publishing updates will be accomplished via [yang-pub-sub-reqts] and [draft-clemm-datastore-push]. Other methods could work equally well .

This is not a new idea. Caching and Content Delivery Networks (CDN) have sped read access for objects within the Internet for years. This has enabled greater performance and scale for certain content. Just as important, these technologies have been employed without end user applications being explicitly aware of their involvement. Such concepts are applicable for scaling the performance of a distributed datastore.

Where caching occurs, it MUST be possible for the Mount Client to store object copies of a remote data node or subtree in such a way that applications are unaware that any caching is occurring. However, the interface to a datastore MAY provide applications with a special mode/flag to allow them to force a read-through.

Where caching occurs, system administration facilities SHOULD allow facilities to flush either the entire cache, or information associated with select Mount Points.

### 5.3. Subscribing to Remote Object Updates

When caching occurs, data can go stale. [draft-clemm-datastore-push] provides a mechanism where changes in an authoritative data node or subtree can be monitored. If changes occur, these changes can be delivered to any subscribing datastores. In this way remote caches can be kept up-to-date. In this way, directly monitoring remote applications can quickly receive notifications without continuous polling.

A Mount Server SHOULD support [draft-clemm-datastore-push] Periodic or On-Change pub/sub capabilities in which one or more remote clients subscribe to updates of a target data node / subtree, which are then automatically published by the Mount Server.

It MUST be possible for Applications to bind to subscribed Data Node / Subtrees so that upon Mount Client receipt of subscribed information, it is immediately passed to the application.

It MUST be possible for a Target Data Node to support 1:n Mount Bindings to many subscribed Mount Points.

### 5.4. Lifecycle of the Mount Topology

Mount can drive a dynamic and richly interconnected mesh of peer-to-peer of object relationships. Each of these Mounts will be independently established by a Mount Client.

It MUST be possible to bootstrap the Mount Client by providing the YANG paths to resources on the Mount Server.

There SHOULD be the ability to add Mount Client bindings during run-time.

A Mount Client MUST be able to be able to create, delete, and timeout Mount Bindings.

Any Subscription MUST be able to inform the Mount Client of an intentional/graceful disconnect.

A Mount Client MUST be able to verify the status of Subscriptions, and drive re-establishment if it has disappeared.

#### 5.4.1. Discovery and Creation of Mount Topology

Application visibility into an ever-changing set of network objects is not trivial. While some applications can be easily configured to know the Devices and available Mount Points of interest, other

applications will have to balance many aspects of dynamic device availability, capabilities, and interconnectedness. For the most part, maintenance of these dynamic elements can be done on the YANG objects themselves without anything needed new for Peer Mount. Technologies such as need reference are covered in other standards initiatives. Therefore this draft does delve deeply into the needs for Auto-discovery of YANG objects which may be advertised.

However it will likely become interesting for a network element to limit the Data Subtrees which might be subscribed for Unsolicited and Periodic Update. It is assumed these capabilities will be included as part of [draft-clemm-datastore-push]

#### 5.4.2. Restrictions on the Mount Topology

Mount Clients MUST NOT create recursive Mount bindings (i.e., the Mount Client should not load any object or subtree which it has already delivered to another in the role of a Mount Server.) Note: Objects mounted from a controller as part of orchestration are *not* considered the same objects as those which might be mounted back from a network device showing the actual running config.

### 5.5. Mount Filter

The Mount Server default MUST be to deliver the same Data Node / Subtree that would have been delivered via direct YANG access.

It SHOULD be possible for a Mount Client to request something less that the full subtree or a target node as defined in [yang-pub-sub-reqts].

### 5.6. Auto-Negotiation of Peer Mount Client QoS

The interest that a Mount Client expresses in a particular subtree SHOULD include the non-functional data delivery requirements (QoS) on the data that is being mounted. Additionally, Mount Servers SHOULD advertise their data delivery capabilities. With this information the Mount Client can decide whether the quality of the delivered data is sufficient to serve applications residing above the Mount Client.

An example here is reliability. A reliable protocol might be overkill for a state that is republished with high frequency. Therefore a Mount Server may sometimes choose to not provide a reliable method of communication for certain objects. It is up to the Mount Client to determine whether what is offered is sufficiently reliable for its application. Only when the Mount Server is offering data delivery QoS better or equal to what is requested, shall a mount binding be established.

Another example is where subscribed objects must be pushed from the Mount Server within a certain interval from when an object change is identified. In such a scenario the interval period of the Mount Server must be equal or smaller than what is requested by a Mount Client. If this "deadline" is not met by the Mount Server the infrastructure MAY take action to notify clients.

## 5.7. Datastore Qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". If on the target node there are multiple datastores available, but there has no specific datastore identified by the Mount Client, then the running or "effective" datastore is the assumed target.

It is conceivable to use such Datastore Qualification in conjunction with ephemeral datastores, to address requirements being worked in the I2RS WG [draft-haas].

## 5.8. Local Mounting

It is conceivable that the mount target does not reside in a remote datastore, but that data nodes in the same datastore as the mountpoint are targeted for mounting. This amounts to introducing an "aliasing" capability in a datastore. While this is not the scenario that is primarily targeted, it is supported and there may be valid use cases for it.

## 5.9. Mount Cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

## 5.10. Transport

Many secured transports are viable assuming transport, data security, scale, and performance objectives are met. Netconf is recommended for starting. Other transports may be proposed over time.

It MUST be possible to support Netconf Transport of subscribed Nodes and Subtrees.

## 5.11. Security Considerations

Many security mechanisms exist to protect data access for CLI and API on network devices. To the degree possible these mechanisms should transparently protect data when performing a Peer Mount.

The same mechanisms used to determine whether a remote host has access to a particular YANG Data Node or Subtree MUST be invoked to determine whether a Mount Client has access to that information.

The same traditional transport level security mechanism security used for YANG over a particular transport MUST be used for the delivery of objects from a Mount Server to a Mount Client.

A Mount Server implementation MUST NOT change any credentials passed by the Mount Client system for any Mount Binding request.

The Mount Server MUST deliver no more objects from a Data Node or Subtree than allowable based on the security credentials provided by the Mount Client.

To ensure the ensuring maximum scale limits, it MUST be possible to for a Mount Server to limit the number of bindings and transactional limits

It SHOULD be possible to prioritize which Mount Binding instances should be serviced first if there is CPU, bandwidth, or other capacity constraints.

## 5.12. High Availability

A key intent for Peer Mount is to allow access to an authoritative copy of an object for a particular domain. Of course system and software failures or scheduled upgrades might mean that the primary copy is not consistently accessible from a single device. In addition, system failovers might mean that the authoritative copy might be housed on a different device than the one where the binding was originally established. Peer Mount architectures must be built to enable Mount Clients to transparently provide access to objects where the authoritative copy moves due to dynamic network reconfigurations .

A Peer Mount architecture MUST guarantee that mount bindings between a Mount Server and Mount Clients are eventually consistent. The infrastructure providing this level of consistency MUST be able to operate in scenarios where a system is (temporarily) not fully connected. Furthermore, Mount Clients MAY have various requirements

on the boundaries under which eventual consistency is allowed to take place. This subject can be decomposed in the following items:

#### 5.12.1. Reliability

Eventual consistency can only be guaranteed when peers are communicating using a reliable method of data delivery. A scenario that deserves attention in particular is when a subset of Mount Clients receive a pushed subscription update. If a Mount Server loses connectivity, cross network element consistency can be lost. In such a scenario Mount Clients MAY elect a new designated Mount Server from the set of Mount Clients which have received the latest state.

#### 5.12.2. Alignment to late joining peers

When a mount binding is established a Mount Server SHOULD provide the Mount Client with the latest state of the requested data. In order to increase availability and fault tolerance an infrastructure MAY support the capability to have multiple alignment sources. In (temporary) absence of a Mount Server, Mount Clients MAY elect a temporary Mount Server to service late joining Mount Clients.

#### 5.12.3. Liveliness

Upon losing liveliness and being unable to refresh cached data provided from a Mount Server, a Mount Client MAY decide to purge the mount bindings of that server. Purging mount bindings under such conditions however makes a system vulnerable to losing network-wide consistency. A Mount Client can take proactive action based on the assumption that the Mount Server is no longer available. When connectivity is only temporarily lost, this assumption could be false for other datastores. This can introduce a potential for decision-making based on semantical disagreement. To properly handle these scenarios, application behavior MUST be designed accordingly and timeouts with regards to liveliness detection MUST be carefully determined.

#### 5.12.4. Merging of datasets

A traditional problem with merging replicated datasets during the failover and recovery of Mount Servers is handling the corresponding target data node lifecycle management. When two replicas of a dataset experienced a prolonged loss of connectivity a merge between the two is required upon re-establishing connectivity. A replica might have been modifying contents of the set, including deletion of objects. A naive merge of the two replicas would discard these

deletes by aligning the now stale, deleted objects to the replica that deleted them.

Authoritative ownership is an elegant solution to this problem since modifications of content can only take place at the owner. Therefore a Mount Client SHOULD, upon reestablishing connectivity with a newly authoritative Mount Server, replace any existing cache contents from a mount binding with the latest version.

#### 5.12.5. Distributed Mount Servers

For selected objects, Mount Bindings SHOULD be allowed to Anycast addresses so that a Distributed Mount Server implementation can transparently provide (a) availability during failure events to Mount Clients, and (b) load balancing on behalf of Mount Clients.

### 5.13. Configuration

At the Mount Client, it MUST be possible for all Mount bindings to configure the following such that the application needs no knowledge. This will include a diverse list of elements such as the YANG URI path to the remote subtree.

### 5.14. Assurance and Monitoring

API usage for YANG should be tracked via existing mechanisms. There is no intent to require additional transaction tracking than would have been provided normally. However there are additional requirements which should allow the state of existing and historical bindings to be provided.

A Mount Client MUST be able to poll a Mount Server for the state of Subsciptions maintained between the two devices.

A Mount Server MUST be able to publish the set of Subscriptions which are currently established on or below any identified data node.

## 6. IANA Considerations

This document makes no request of IANA.

## 7. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ambika Prasad Tripathy. Shashi Kumar Bansal, Prabhakara Yellai, Dinkar Kunjikrishnan, Harish Gumaste, Rohit M., Shruthi V. , Sudarshan Ganapathi, and Swaroop Shastri.

# 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.

[RFC3768] Hinden, R., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, April 2004.

[RFC4610] Farinacci, D. and Y. Cai, "Anycast-RP Using Protocol Independent Multicast (PIM)", RFC 4610, August 2006.

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

## 8.2. Informative References

[ICCP] Martini, Luca., "Inter-Chassis Communication Protocol for L2VPN PE Redundancy", March 2014, <https://tools.ietf.org/html/draft-ietf-pwe3-iccp-16>.

[OMG-DDS] "Data Distribution Service for Real-time Systems, version 1.2", January 2007, <http://www.omg.org/spec/DDS/1.2/>.

[draft-clemm-datastore-push]
Clemm, Alex., "Subscribing to datastore push updates", March 2015.

[draft-clemm-mount]
Clemm, Alex., "Mounting YANG-Defined Information from Remote Datastores", October 2014, <http://tools.ietf.org/id/draft-clemm-netmod-mount-02.txt>.

[draft-haas]
Haas, J., "I2RS requirements for netmod/netconf draft-haas-i2rs-netmod-netconf-requirements-00", September 2014, <draft-haas-i2rs-netmod-netconf-requirements>.

[rfc6020bis]
Bjorklund, Martin., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", January 2015, <https://tools.ietf.org/html/draft-ietf-netmod-rfc6020bis-03>.

[yang-pub-sub-reqts]
Voit, Eric., Clemm, Alex., and Alberto. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", March 2015.

## 8.3. URIs

[1] http://thomaswdinsmore.com/2014/05/01/distributed-analytics-primer/

[2] http://en.wikipedia.org/wiki/ACID

[3] http://robertgreiner.com/2014/08/cap-theorem-revisited/

[4] http://guide.couchdb.org/draft/consistency.html

## Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alex Clemm
Cisco Systems

Email: alex@cisco.com

Sander Mertens
Prismtech

Email: sander.mertens@prismtech.com