Network Working Group A. Choudhary
Internet-Draft S. Shah
Intended status: Standards Track Cisco Systems
Expires: April 25, 2015 October 22, 2014

YANG Model for Diffserv
draft-asechoud-netmod-diffserv-model-00

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 April 25, 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
         +--ro classifier-entry-statistics
            +--ro classified-pkts?    uint64
            +--ro classified-bytes?   uint64
            +--ro classified-rate?    uint64
```

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
    +--rw policy-entry* [policy-name]
       +--rw policy-name         string
       +--rw policy-descr?       string
       +--rw classifier-entry* [classifier-entry-name]
          +--rw classifier-entry-name           leafref
          +--rw classifier-entry-inline?        boolean
          +--rw classifier-entry-filter-oper?   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
          +--rw classifier-action-entry-cfg* [action-type]
          |  +--rw action-type    identityref
          |  +--rw (action-cfg-params)?
          +--rw child-policy?                   leafref
```

ietf-diffserv-action module contains set of diffserv actions which are augmented to diffserv-policy module. Marking sets Diffserv codepoint value in the classified packet. Color-aware and Color-

blind meters can be configured. Action counters are also augmented to diffserv-policy module.

```
module: ietf-diffserv-action
augment /diffserv-policy:policies/diffserv-policy:policy-entry/
         diffserv-policy:classifier-entry/
         diffserv-policy:classifier-action-entry-cfg/
         diffserv-policy:action-cfg-params:

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

```
      +--:(algorithmic-drop)
      |  +--rw (drop-algorithm)?
      |     +--:(always-drop)
      |     |  +--rw drop-cfg
      |     |     +--rw drop-action?   boolean
      |     +--:(tail-drop)
      |     |  +--rw tail-drop-cfg
      |     |     +--rw qlimit-dscp-thresh* [dscp-min dscp-max]
      |     |        +--rw dscp-min     inet:dscp
      |     |        +--rw dscp-max     inet:dscp
      |     |        +--rw threshold
      |     |           +--rw (threshold-type)?
      |     |              +--:(size)
      |     |              |  +--rw threshold-size?       uint64
      |     |              +--:(interval)
      |     |                 +--rw threshold-interval?   uint64
      |     +--:(head-drop)
      |     |  +--rw head-drop-cfg
      |     |     +--rw qlimit-dscp-thresh* [dscp-min dscp-max]
      |     |        +--rw dscp-min     inet:dscp
      |     |        +--rw dscp-max     inet:dscp
      |     |        +--rw threshold
      |     |           +--rw (threshold-type)?
      |     |              +--:(size)
      |     |              |  +--rw threshold-size?       uint64
      |     |              +--:(interval)
      |     |                 +--rw threshold-interval?   uint64
      |     +--:(random-detect)
      |        +--rw random-detect-cfg
      |           +--rw exp-weighting-const?   uint32
      |           +--rw mode-aggregate?        boolean
      |           +--rw wred-dscp-thresh* [dscp-min dscp-max]
      |              +--rw dscp-min            inet:dscp
      |              +--rw dscp-max            inet:dscp
      |              +--rw wred-min-thresh
      |              |  +--rw threshold
      |              |     +--rw (threshold-type)?
      |              |        +--:(size)
      |              |        |  +--rw threshold-size?       uint64
      |              |        +--:(interval)
      |              |           +--rw threshold-interval?   uint64
      |              +--rw wred-max-thresh
      |              |  +--rw threshold
      |              |     +--rw (threshold-type)?
      |              |        +--:(size)
      |              |        |  +--rw threshold-size?       uint64
      |              |        +--:(interval)
      |              |           +--rw threshold-interval?   uint64
```

```
      |              +--rw mark-probability?   uint32
      +--:(min-rate)
         +--rw min-rate-cfg
            +--rw min-rate?   uint64

   augment /diffserv-policy:policies/diffserv-policy:policy-entry/
           diffserv-policy:classifier-entry:
      +--rw queuing-statistics
         +--ro output-pkts?        uint64
         +--ro output-bytes?       uint64
         +--ro queue-size-pkts?    uint64
         +--ro queue-size-bytes?   uint64
         +--ro drop-pkts?          uint64
         +--ro drop-bytes?         uint64
         +--rw wred-statistics* [dscp-min dscp-max]
            +--rw dscp-min            inet:dscp
            +--rw dscp-max            inet:dscp
            +--ro early-drop-pkts?    uint64
            +--ro early-drop-bytes?   uint64
```

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]
         +--rw policy-name?    string
         +--rw policy-descr?   string
         +--rw direction       identityref
```

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 2014-10-07 {
       description
         "First revision of diffserv based classifier";
     }

     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 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";
```

```
        }
      }
    }

    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 {
        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;
        }
        container classifier-entry-statistics {
          config false;
          description
            "
             This group defines the classifier filter statistics of
             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";
          }
        }
      }
    }
  }
```

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 2014-10-07 {
      description
        "First revision of diffserv policy";
    }

    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";
    }

    container policies {
      description
        "list of policy templates";
      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 leafref {
              path "/classifier:classifiers/classifier:classifier-entry/
                    classifier:classifier-entry-name";
            }
          }
          uses classifier:classifier-entry-inline-attr;
          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;
          }
          leaf child-policy {
            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-inet-types {
      prefix inet;
    }
    import ietf-diffserv-classifier {
      prefix diffserv-classifier;
    }
    import ietf-diffserv-policy {
      prefix diffserv-policy;
    }

    revision 2014-10-07 {
      description
        "Initial revision for diffserv actions on network packets";
    }

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

    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;
      uses meter-action-statistics;
    }

    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 {
      uses leaky-bucket;
    }

    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
           the 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
          ";
      }
      list wred-dscp-thresh {
        key "dscp-min dscp-max";
        uses dscp-range;
        uses wred-threshold;
      }
    }

    grouping meter-action-statistics {
      description
        "Meter statistics";
      leaf metered-pkts {
        type uint64;
        config false;
        description
          "Number of packets counted by the meter";
      }
      leaf metered-bytes {
        type uint64;
        config false;
        description
          "Bytes of packets counted by the meter";
      }
      leaf metered-rate {
        units "bits-per-second";
        type uint64;
        config false;
        description
          "Traffic Rate measured by the meter";
      }
    }

    grouping wred-class-counts {
      leaf early-drop-pkts {
        type uint64;
        config false;
        description
```

```
          "Early drop packets ";
      }
      leaf early-drop-bytes {
        type uint64;
        config false;
        description
          "Early drop bytes ";
      }
    }

    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 head-drop {
            container head-drop-cfg {
              uses queuelimit;
            }
          }
          case random-detect {
            container random-detect-cfg {
              uses randomdetect;
            }
          }
        }
      }
      case min-rate {
        container min-rate-cfg {
          uses min-rate;
        }
      }
    }
    augment "/diffserv-policy:policies/diffserv-policy:policy-entry/
             diffserv-policy:classifier-entry" {
      container queuing-statistics {
        description
          "queue related statistics ";
        leaf output-pkts {
          type uint64;
          config false;
          description
            "Number of packets transmitted from queue ";
        }
        leaf output-bytes {
          type uint64;
          config false;
          description
            "Number of bytes transmitted from queue ";
        }
        leaf queue-size-pkts {
          type uint64;
          config false;
          description
            "Number of packets currently buffered ";
        }
        leaf queue-size-bytes {
          type uint64;
          config false;
          description
            "Number of bytes currently buffered ";
```

```
        }
        leaf drop-pkts {
          type uint64;
          config false;
          description
            "Total number of packets dropped ";
        }
        leaf drop-bytes {
          type uint64;
          config false;
          description
            "Total number of bytes dropped ";
        }
        list wred-statistics {
          key "dscp-min dscp-max";
          description
            "WRED statistics for a dscp range ";
          uses dscp-range;
          uses wred-class-counts;
        }
      }
    }
  }
```

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-policy {
      prefix policy;
    }

    revision 2014-10-07 {
      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";
    }

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

    augment "/if:interfaces/if:interface" {
      list diffserv-target-entry {
        key "direction";
        description
          "policy target for inbound or outbound direction";
        uses policy-target-generic-attr;
      }
    }
  }
```

6. Security Considerations

7. Acknowledgement

The editor of this document wishes to thank Fred Baker for overviewing the document and provide useful comments.

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.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 28, 2015

A. Bierman
YumaWorks

September 24, 2014

YANG Conformance Specification
draft-bierman-netmod-yang-conformance-04

Abstract

This document describes conformance specification and advertisement mechanisms for NETCONF servers implementing YANG data model modules.

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 March 28, 2015.

Copyright Notice

Table of Contents

1. Introduction

There is a need for standard mechanisms to allow YANG [RFC6020] data model designers to express more precise and robust conformance levels for server implementations of a particular YANG module, or set of YANG modules.

There is also a need for standard mechanisms to allow NETCONF [RFC6241] servers to precisely advertise the conformance level of each YANG module it supports.

This document describes some problems with the current conformance specifications mechanisms in YANG and conformance advertisement mechanisms in NETCONF. Solution proposals are also presented to address these problems.

1.1. Terminology

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].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- capability
- client
- datastore
- protocol operation
- server

1.1.2. YANG

The following terms are defined in [RFC6020]:

- data node
- extension
- feature

- grouping
- identity
- module
- notification
- submodule
- typedef

1.1.3. Terms

The following terms are used within this document:

- base module: There is an implied "base" subset of a YANG module, which includes all "rpc", "data-def", and "notification" statements which are not conditional. The base module may be empty, a subset of all statements, or the entire module.
- conditional node: An object that has one or more "if-feature" sub-statements associated with it. Note that objects affected by "when" statements are not considered conditional for conformance purposes.
- import-by-revision: A YANG import statement that includes a revision-date statement. This specifies the exact revision of the YANG module to import, instead of the server picking the revision to import.
- include-by-revision: A YANG include statement that includes a revision-date statement. This specifies the exact revision of the YANG submodule to include, instead of the server picking the revision to include.
- object: a conceptual data structure represented by a YANG data, rpc, or notification statement.
- revision identifier: a YANG module revision is identified by the revision date value of the most recent revision statement in the module. If there are no revision statements then the empty string is the revision identifier.
- schema tree: The conceptual tree of all objects derived from the set of all YANG modules supported by the server. This tree only includes conditional nodes if all corresponding if-feature statements are "true". Any deviation statements have also been

conceptually applied to the schema tree as well.

- YANG dependency tree: A conceptual tree containing the set of all revision identifiers for all modules and submodules in the schema tree.

1.1.4. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Overview

The YANG module represents a conceptual API contract between the client and server for a management protocol that uses YANG. There are two primary factors that directly impact interoperability.

- Client and Server Implementation Requirements
- Actual Server Implementation Capabilities

The syntax and semantics of a given data model, used within a specific protocol, must be understood by the client and server developers to achieve interoperability.

These implementation requirements need to stable, and not change for a given revision of a module. Any changes to the implementation requirements needs to be intentional, and properly identified and understood by client and server developers.

2.1. Server Implementation Capabilities

The actual implemented portions of a given data model, used within a specific protocol, must be understood by the client. For YANG, a "YANG Module capability URI" is advertised in the server capabilities, and understood by the client.

For improved interoperability, the server must identify its conformance level for each module, and identify and deviations from the advertised conformance level.

3. Problems With YANG Conformance Mechanisms

This section describes some perceived deficiencies with the current YANG data model conformance specification and NETCONF server conformance advertisement mechanisms.

The engineering trade-off that needs to be considered is the complexity and effort required to specify conformance information and the accuracy and specificity of the conformance information that is defined.

The rest of this section discusses the advantages and disadvantages of the two solution paths available in YANG to identify conformance requirements.

Option 1: do not use import-by-revision or include-by-revision at all. This approach requires some other mechanism besides the import and include statements to fully identity the dependency tree.

Option 2: use import-by-revision and include-by-revision everywhere so the YANG dependency tree is fully specified.

3.1. Incomplete Dependency Tree: Conformance Drift

If the import-by-revision and include-by-revision mechanisms are not used everywhere in the dependency tree, then the application compiling the YANG modules must decide somehow which revision to use for any module or submodule without a revision-date statement.

When new modules are added which require the revision of a shared module to be updated, the server will need to advertise the updated module. There is ambiguity whether the server needs to advertise all the revisions of all the YANG modules is uses, and what that even means in some cases.

Conformance drift occurs when definition updates in an imported module inadvertently change the syntax and/or semantics of statements in the importing module. These changes occur without a change in the revision date for the importing module, which makes them hidden undocumented conformance updates.

3.1.1. Typedef Drift

Typedef drift can occur if a leaf or leaf-list "type" statement references a "typedef" statement from an external module. The YANG syntax specified for a leaf or leaf-list will depend on which revision of the external module is used. If the newer revision is used, then the allowed value set will be changed.

There are several ways this is allowed under the conditions described in section 10 of [RFC6020]:

- an "enumeration" type can add new "enum" statements
- a "bits" type can add new "bit" statements
- a "range", "length", or "pattern" statement may expand the allowed value space

Note that conformance for "identityref" types are discussed separately. This data type is complex because the value set is distributed, and actual allowed values of this type are node and/or context-specific.

3.1.2. Grouping Drift

Grouping drift can occur if a "uses" statement references a "grouping" statement from an external module, and the grouping contents change. New data nodes can be added to a grouping in a new revision of a module under certain conditions, described in section 10 of [RFC6020]:

The YANG data nodes that are copied from the grouping will depend on which revision of the external module is used. If the newer revision is used, then any new and/or modified objects will be added to the schema tree.

It is not clear how multiple revisions of the same grouping can be used reliably. It depends on the content of the grouping. For example, a grouping may contain objects with "must" or "when" XPath expressions. These expressions can reference objects inside or outside the grouping in a manner that has the desired effect when evaluated with the intended revision of the grouping, but incorrect effect when evaluated against newer modules.

It is therefore unsafe for a server to support multiple revisions of a grouping, in a generic manner.

3.1.3. Conformance Drift Example

Consider the "first release" where the module set advertised by the server consists of 2 modules:

```
    - module A, revision 2014-01-01
    - module B, revision 2014-01-02

     module A {
       namespace "module-A";
       prefix A;
       revision "2014-01-01" {
         description "First revision";
       }

       typedef knob-range {
         type int32 { range "1 .. 100"; }
       }

       grouping knob-group {
         leaf knobA {
           type knob-range;
         }
       }

       leaf which-leaf {
         type knob-range;
       }
     }

     module B {
       namespace "module-B";
       prefix B;
       import A { prefix A; }
       revision "2014-01-02" {
         description "First revision";
       }

       leaf knob1 {
         type A:knob-range;
       }

       container knobs {
         uses A:knob-group;
       }
     }
```

The server can clearly express the value range allowed for "knob1" in revision "2014-01-02" of the "B" module. There is only 1 leaf ("knobA") in the "knobs" container. There is only 1 possible revision of module "A" that can be advertised by a server, so the definition of the "which-leaf" leaf is clear.

Now consider the "second release" where the module set has been extended to 3 modules, and module "A" has been updated but module "B" has not been updated. The server advertises the latest revision of module "A" instead of both revisions. This module was updated to support module "C".

```
    - module A, revision 2014-02-01 (updated)
    - module B, revision 2014-01-02 (unchanged)
    - module C, revision 2014-02-02 (new)


     module A {
       namespace "module-A";
       prefix A;
       revision "2014-02-01" {
         description "Second revision";
       }
       revision "2014-01-01" {
         description "First revision";
       }

       typedef knob-range {   // range expanded and default added!
         type int32 { range "1 .. 500"; }
         default 500;
       }

       grouping knob-group {
         leaf knobA {
           type knob-range;
         }
         leaf knobB {               // added 1 leaf to grouping!
           type knob-range;
         }
       }

       leaf which-leaf {
         type knob-range;          // type changed!
         default 200;              // added default!
       }
     }

     module C {
       namespace "module-C";
       prefix C;
       import A { prefix A; }
       revision "2014-02-02" {
         description "First revision";
       }

       leaf knob2 {
         type A:knob-range;
       }
     }
```

There are unadvertised changes in module "B":

- "knob1" leaf range expanded and default added

- "knobB" leaf added to the "knobs" container

Problems can occur if the client understanding of the dependency tree is not accurate. It might assume the server has updated the instrumentation for the "knob1" leaf so the increased range and new default are supported. The "knob1" syntax used depends on which revision of module "A" that is parsed.

Problems can occur if the server instrumentation understanding of the dependency tree is not accurate. It might assume the server validation code will only give it a valid value for leaf "knob1" which fits in one byte. Values from 256 to 500 in this example might cause internal errors in the server.

If the server advertises the latest revisions of the YANG modules then syntax of "knob1" and contents of the "knobs" container will change, even though module "B" has not been updated. This can cause the client to send invalid protocol requests or misinterpret data received from the server.

3.2. Complete Dependency Tree: Multiple Concurrent Revisions

If exact revision dates are used everywhere then they need to be carefully examined each time any module in the dependency tree is changed. Updating modules just to change the revision date of an updated imported module also changes the revision date of the importing module. This ripple effect creates many revisions that do not even import the particular module, and would add significant complexity to module lifecycle management.

If some dependencies are updated but not others, then the client and server will need to support multiple revisions of the same module or submodule at the same time. This is the most likely scenario to occur.

NETCONF and YANG do not specify how a server must support multiple revisions of the same module at the same time. This causes interoperability problems which need to be addressed.

Consider the previous example, except import-by-revision is used everywhere to fix the conformance drift problem. The only changes at all are the addition of revision dates in the import statements in modules "B" and "C".

3.2.1. Conformance Ambiguity Example

If import-by-revision is used everywhere, then the second release in the previous example would contain the following modules and revisions advertised by the server:

- module A, revision 2014-01-01
- module A, revision 2014-02-01
- module B, revision 2014-01-02
- module C, revision 2014-02-02

The conformance drift problems in module "B" are fixed. Module "B" use the old definitions from the first revision of module "A", and module "C" uses the new definitions from the updated revision.

However, now that 2 revisions of module "A" are advertised by the server, it is not clear what revision of leaf "which-leaf" from module "A" is implemented by the server.

This problem applies to all protocol accessible statements (data nodes, "rpc", and "notification" statements). It may also apply to any global reusable statements such as "extension" and "feature". Advertising multiple revisions of any of these statements causes ambiguity in the conformance definition.

3.2.2. Augmenting External Data Nodes

The problem of ambiguous conformance affects data nodes derived from external augmentation, i.e., several modules (all using import-by-revision) augment multiple revisions of the same data nodes. The augmenting nodes can change over time, and they can themselves contain imported definitions.

The YANG conformance rules do not support different revisions of the same data node instance. The actual implementation matrix cannot be an ad-hoc subset of all possible revision combinations. The client needs to be able to identify the exact revisions of data nodes that are supported by the server.

3.2.3. Identityref Value Sets

A server cannot advertise even the complete set of identities that it supports. It actually advertises all of the identities in all modules in the dependency tree. This is a superset of all supported identities by the server.

It is not possible for a client to determine the supported identity set at all. In addition, individual identityref leafs may support

different subsets of all possible identities supported by the server. For example, simply importing the "iana-if-types" YANG module does not mean the server supports every possible interface type that has ever been defined.

YANG conformance requirements for "identityref" leafs are unclear and need to be clarified. Discovery of agent capabilities of actual supported identities is needed.

3.2.4. Leafref Value Sets

YANG conformance requirements for "leafref" leafs are unclear and need to be clarified. Support for discovery of agent capabilities of actual supported identities is needed. The same issues that apply to "identityref" types can occur with "leafref" types.

3.3. Module Capability Advertisement Issues

NETCONF servers advertise the YANG modules they support as <capability> URI strings in the <hello> message. The complete list of modules used by the server needs to be advertised in order for the client application to correctly parse the YANG modules and reproduce the schema tree used by the server. However the client does not really know which modules are advertised for full conformance, and which are advertised for partial conformance (such as importing typedef and identity statements from the module). The conformance information that is derived from the YANG module advertisement needs to be clarified.

4. YANG Conformance Guidelines

Conformance for the "ietf-yang-conformance" module capability requires implementation of these conformance guidelines.

4.1. Conformance is Based on the Module Set, not One Module

The reason conformance drift and conformance ambiguity are currently problems is due to the incorrect assumption that a YANG datastore is allowed to be considered a collection of independent module implementations. This is done in pursuit of module independence and the ability to support off-line validation tools of YANG content.

The "individual module" approach does not work for YANG datastores. Instead, the YANG conformance needs to be based on the entire module set, identified by the collection of YANG modules and submodules, features and deviations that the server is using at the moment.

The revision date for a specific module or submodule only freezes the definitions within that module or submodule. Any external modules or submodules have their own revision date, that cannot be frozen in the server implementation.

It is the responsibility of the YANG module writer to ensure that a new or existing module is syntactically and semantically compatible with the current revision of all external modules imported by that module. It is the responsibility of the NETCONF server developer to ensure that the supported module set is syntactically and semantically self-compatible.

4.2. Import has no Conformance Semantics

Simply importing a module implies no conformance relationship between the importing and imported module. There are many ways that YANG can be used so an imported module is not even used. E.g., the only identifiers from the imported module have "if-feature" statements, but the server does not enable any of the features. The "import" statement is only useful for resolving external identifiers used in the current module.

4.3. Only the Most Recent Revision Allowed

The server MUST implement the most recent revision of each module it advertises. The vendor cannot upgrade a shared module without updating all the modules that actually depend on the changes.

If any of the following statements in the most recent advertised revision of the base module are not supported by the server, then a

YANG module containing the appropriate "deviation" statements MUST be advertised for that module.

- o augment-stmt
- o data-def-stmt
- o notification-stmt
- o rpc-stmt

If a shared module needs to be updated, then the conformance is implicitly updated for the modules that depend on the changes.

4.3.1. YANG Module Capability URIs

There are existing NETCONF client applications that assume that all the YANG modules used by the server will be sent in the <hello> message. These applications are likely to fail if an incomplete module set is advertised by the server.

The server MUST advertise the most recent revision of all YANG modules that it supports, including modules that do not contain any data definitions.

If the server uses any modules with submodules, then the "submodule" capability in Section 5 SHOULD be advertised for each submodule used. Only one revision of each submodule SHOULD be advertised.

4.4. Import and Include By Revision Do Not Really Help

The simplest solution for fixing the conformance drift problem would be to use import and include by revision everywhere, and make this usage mandatory. Any new YANG statements would cause duplication of the module or submodule name and revision date information. However, this solution is not workable because the ripple effect will require constant updating of many YANG modules just to change the revision-date clause.

Multiple concurrent revisions of YANG datastore contents are not supported in a NETCONF server. The server behavior required for import-by-revision has never been specified wrt/ multiple revisions of the same module.

The "revision-date" statement SHOULD NOT be used in "import" statements. Conformance for external modules cannot be fixed to a specific revision when used within a NETCONF server. If the "revision-date" statement is used, it SHOULD be interpreted as "the

current revision of the imported module when this module was published". The actual revision used by the server will be advertised in the YANG module capability URI.

The hard-wired revision dates in "import" statements are only useful for off-line validation. of protocol requests. Even then, they must be used everywhere to be deterministic. Real NETCONF sessions require not only all the module names and revision dates, but the enabled YANG features and YANG conformance deviations.

4.5. Limit Protocol Accessible Objects in the Base Module

YANG module designers SHOULD NOT specify "augment", "rpc", "notification", or any data definition statements in the base module of a YANG module, if it also contains any "typedef", "grouping", "identity", "extension", or "deviation" statements. This will prevent the need for deviations advertisement by the server, just to utilize the definitions designed for reusability.

Note that the "feature" statement is not included in this list, otherwise if any optional protocol-accessible statements were needed, then they would all need to be optional.

Reusable definitions can be placed in a separate module with no protocol-accessible statements, or the protocol-accessible statements can contain "if-feature" statements to remove them from the base module. Either practice is acceptable.

5. YANG Submodule Capability

5.1. Overview

A new capability called "submodule" is defined to identify the specific revisions of all submodules the server is using.

When a new session is started, the client can examine the "submodule" <capability> URIs sent by the server to determine the specific revision that the server is using for each submodule.

Since submodule names share the same namespace as module names, there is no need to add the parent module name and revision to the advertisement. The server is required to support the advertised revision, and only one revision can be advertised.

5.1.1. :submodule Capability Example

The following example show just the "submodule" capability for submodule "submod-A" with revision date "2014-09-23", and submodule "submod-B" with revision date "2013-10-01", The URIs are wrapped for display purposes only.

```
  <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <capabilities>
    <capability>
     urn:ietf:params:netconf:capability:submodule:1.0?
        name=submod-A&revision=2014-09-23
    </capability>
    <capability>
     urn:ietf:params:netconf:capability:submodule:1.0?
        name=submod-B&revision=2013-10-01
    </capability>
    // ... rest of <capability> elements
   <capabilities>
   <session-id>3</session-id>
  </hello>
```

5.2. Dependencies

The :submodule capability is not dependent on any other capabilities.

5.3. Capability Identifier

The :submodule capability is identified by the following capability string:

urn:ietf:params:netconf:capability:submodule:1.0

The :submodule capability SHOULD be sent in every server <hello> message, for each submodule used by the server. The "submodule" parameter for the :submodule capability MUST be present, and is set to the name of the submodule. The "revision" parameter for the :submodule capability MUST be present, and is set to the revision date of the submodule.

5.4. New Operations

The :submodule capability does not introduce any new protocol operations.

5.5. Modifications to Existing Operations

The :submodule capability does not modify any existing protocol operations.

5.6. Interactions with Other Capabilities

The :submodule capability does not interact with any other capabilities.

6. YANG Conformance Module

The "ietf-yang-conformance" module in Section 6.4 defines 2 protocol operations for retrieving allowed value sets from the server. There is also an augmentation of the "schema" list in the "ietf-netconf-monitoring" module defined in [RFC6022].

```
  augment /ncm:netconf-state/ncm:schemas/ncm:schema:
     +--ro conformance-type?    enumeration
     +--ro belongs-to-module?   string
  rpcs:
     +---x get-allowed-identities     {get-allowed}?
     |  +--ro input
     |  |  +--ro target    instance-identifier
     |  +--ro output
     |     +--ro (response)?
     |     |  +--:(all)
     |     |  |  +--ro all?           empty
     |     |  +--:(identity)
     |     |     +--ro identity*      string
     |     +--ro support-all?   empty
     +---x get-allowed-leafrefs       {get-allowed}?
        +--ro input
        |  +--ro target    instance-identifier
        +--ro output
           +--ro (response)?
           |  +--:(all)
           |  |  +--ro all?           empty
           |  +--:(value)
           |     +--ro value*         string
           +--ro support-all?   empty
```

6.1. Identityref Value Set Discovery

A server is not required to support all advertised identity values. The actual values required for a particular "identityref" leaf or leaf-list MAY be specified in the YANG module. If not, then the entire list of identities with a matching "base" definition is used as the superset of all allowed values for the identityref.

A new protocol operation called "get-allowed-identities" is defined to allow a client to discover the set of identities actually supported at the moment for a specific leaf or leaf-list.

6.2. Leafref Value Set Discovery

A server is not required to support all existing potential instances for a leaf or leaf-list using the "leafref" data type. The entire

list of instances of the data node specified in the "path" statement is used as the superset of all allowed values for the leafref type.

A new protocol operation called "get-allowed-leafrefs" is defined to allow a client to discover the set of leafref values actually supported at the moment for a specific leaf or leaf-list.

6.3. Schema Conformance Discovery

The "schema" list in the "ietf-netconf-monitoring" module identifies all the YANG modules available to the server. A new leaf called "conformance-type" is defined to augment this list.

Two conformance types are defined, called "base" and "reuse". This information allows clients to understand how each module is used, without parsing and and analyzing the "capability" URIs from the "ietf-netconf-monitoring" module.

6.4. YANG module

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-yang-conformance@2014-09-24.yang"

```
module ietf-yang-conformance {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-conformance";
  prefix "yangconf";
  import ietf-netconf-monitoring { prefix ncm; }

  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:   Andy Bierman
               <mailto:andy@yumaworks.com>";

  description
    "This module contains RPC statements to identify
```

```
        the server capabilities for specific data nodes.
        NETCONF Monitoring extensions are also provided
        to report YANG conformance information.

        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.";

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

    // RFC Ed.: remove this note
    // Note: extracted from draft-bierman-netmod-restconf-01.txt

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2014-09-24 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Conformance.";
    }

    feature netconf-monitoring {
      description
        "Indicates the ietf-netconf-monitoring extensions
         defined in this module are supported.";
    }

    feature get-allowed {
      description
        "Indicates the 'get-allowed-identities' and
         'get-allowed-leafrefs' protocol operations are supported.";
    }

    rpc get-allowed-identities {
      if-feature get-allowed;
      description
        "Returns the set of identity values that are supported for
```

```
          a specific data node instance.  The server will indicate
          if all instances of the specified data node accept the
          same value set.

          The server will return an error with an
          'invalid-value' error-tag if the target parameter
          referenced an invalid instance.

          The server MAY support this operation for operational
          data (i.e. config false). If not, then an error with an
          'operation-not-supported' error-tag will be returned
          by the server if the target parameter does not identify
          a data node in the configuration datastore";

       input {
         leaf target {
           type instance-identifier {
             require-instance false;
           }
           mandatory true;
           description
             "Identifies an instance or possible instance of
              a data node that uses the identityref data type.
              The server will return information about the
              accepted values for this data node.";
         }
       }
       output {
         choice response {
           description
             "Either the 'all' element is returned or one or more
              'identity' elements are returned in the response.";

           leaf all {
             type empty;
             description
               "Indicates all advertised identities with a matching
                base value are supported for the specified data node.";
           }
           leaf-list identity {
             type string;
             description
               "This formatted string contains the value of an
                acceptable identity for the specified identityref
                data node. It has the following format:

                  <module-name>:<identity-name>
```

```
                The module name is followed by the colon (':')
                character, which is followed by the identity name.
                No whitespace is allowed.";
            }
          }
          leaf support-all {
            type empty;
            description
              "Indicates all possible instances of the target data node
               support the same identities.";
          }
        }
      }

     rpc get-allowed-leafrefs {
       if-feature get-allowed;
       description
         "Returns the set of leafref values that are supported for
          a specific data node instance.  The server will indicate
          if all instances of the specified data node accept the
          same value set.

          The server will return an error with an
          'invalid-value' error-tag if the target parameter
          referenced an invalid instance.

          The server MAY support this operation for operational
          data (i.e. config false). If not, then an error with an
          'operation-not-supported' error-tag will be returned
          by the server if the target parameter does not identify
          a data node in the configuration datastore";

       input {
         leaf target {
           type instance-identifier {
             require-instance false;
           }
           mandatory true;
           description
             "Identifies an instance or possible instance of
              a data node that uses the leafref data type.
              The server will return information about the
              accepted values for this data node.";
         }
       }
       output {
         choice response {
           description
```

```
            "Either the 'all' element is returned or one or more
             'value' elements are returned in the response.";
          leaf all {
            type empty;
            description
              "Indicates all current instances of the data node
               identified by the 'path' statement for the
               target parameter are supported.";
          }
          leaf-list value {
            type string;
            description
              "This string contains the value of an
               acceptable leafref, encoded as a string.";
          }
        }
        leaf support-all {
          type empty;
          description
            "Indicates all possible instances of the target data
             node support the same leaf values.";
        }
      }
    }

    augment /ncm:netconf-state/ncm:schemas/ncm:schema {
      if-feature netconf-monitoring;
      description
        "Conformance information added to each 'schema' list entry.";

      leaf conformance-type {
        type enumeration {
          enum base {
            description
              "Indicates the entire base module is supported within
               the server. The server MAY also advertise YANG features
               and YANG deviations for this module.";
          }
          enum reuse {
            description
              "Indicates that no data definition, augment, rpc,
               or notification statements in the base module are
               used from the module. Only reusable definitions are
               used, which include the following statements:
                - extension-stmt
                - feature-stmt
                - identity-stmt
                - typedef-stmt
```

```
                  - grouping-stmt
                ";
            }
          }
          config false;
          description
            "Indicates how the server is claiming conformance
             for the module represented by this schema.";
        }

        leaf belongs-to-module {
          type string;
          config false;
          description
            "This leaf is only applicable if the schema identified
             this entry represents a YANG submodule.  If so, then this
             string identifies the parent module name for the
             submodule.";
        }

      }


   }
```

<CODE ENDS>

7. IANA Considerations

7.1. YANG Module Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-conformance
Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

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

```
name:         ietf-yang-conformance
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-conformance
prefix:       yangconf
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX
```

8. Security Considerations

TBD.

Access control may need to be considered. For example, a server may wish to prune values returned from the "get-allowed-identities" or "get-allowed-leafrefs" operations, based on the client user identity.

9. Open Issues

9.1. YANG Module Advertisement

9.1.1. Capability-ID vs. Module URI List

Left out of this release:

```
   augment /ncm:netconf-state/ncm:capabilities {
     if-feature netconf-monitoring;

     leaf capability-set-id {
       config false;
       type string;
       description
        "Contains the capability set identifier for the
         current set of capability URIs.  The server MUST
         update this value if any capability URIS are added,
         modified, or deleted.

         The server SHOULD select a previously unused value
         each time the value is changed.

         This object can enable caching of the capability URI
         list by a client.";
     }
   }
```

The "capability-id" was first introduced in "draft-bierman-netconf-efficiency-extensions-00.txt" in October 2013. Switching from <hello> based module set discovery to retrieval-based module set discovery requires a "flag day" upgrade, and the data that is replacing the removed <capability> URIs needs to be mandatory-to-implement.

If the <hello> is really too large, then a single <capability-set-id> URI could be used instead, so the client could cache the data and reduce bandwidth (at the cost of maintaining a cache). It is not clear that the <hello> message is really too large relative to SSH setup and other data.

9.1.2. Full vs. Partial <capability> List

If only supported modules are advertised in the <hello> then the client has an incomplete list and must retrieve the revision dates of all the modules that are just imported. If the client has to retrieve any information at all to complete the module set, then it might as well retrieve all of it.

9.2. YANG 1.1 Support

The YANG 1.1 plan for server capability discovery is to only advertise YANG 1.0 modules, and not advertise any YANG 1.1 modules. Instead, a hash or instance-identifier will be used for client caching, and the client will retrieve the module information instead. It is not known if any new mechanisms will be needed in this document.

9.3. Protocol Independence

- Should this document be written so it is specific to the NETCONF and RESTCONF protocols?

- If protocol independence is required, then what parts of the draft need to be changed? What parts need to be moved to a different document?

10. Change Log

-- RFC Ed.: remove this section before publication.

10.1. 03-04

- Massive rewrite to address just the issues of conformance drift and conformance ambiguity within a single YANG module.
- Removed package definitions and service profile definitions. Deferred for future work.
- Added ietf-yang-conformance YANG module
- changed term "module base" to "base module"
- Removed term "YANG feature set"
- Added terms "import-by-revision", "include-by-revision", "revision-identifier"
- Added :submodule capability

10.2. 02-03

- updated routing example

10.3. 01-02

- removed ’min-revision’ and ’max-revision’ statements.
- added mandatory revision-stmt for profiles and require-package statements.
- changed package capability to allow the server to advertise multiple profiles for a YANG package instead of 1.
- changed required package contents from 0 to 1.
- removed ’category’ statement.
- add ’require-parameter’ statement to require-capability-stmt.
- add ’require-value’ statement to require-parameter-stmt.
- removed ’prefix-stmt’ from YANG package.

- removed ’header-stmts’ from ABNF so prefix-stmt would be removed.
- added ’yangconf-version-stmt’ to ABNF, cloned from RFC 6020
- added ’namespace-stmt’ to ABNF, cloned from RFC 6020
- remove conformance profile ’ip’ from example, since this is now achieved by advertising multiple names in the ’profiles’ parameter in the ’package’ capability URI.

10.4. 00-01

- fixed typos in text and examples
- updated ietf-routing-pkg example
- added ’require parameters for capabilities’ as an open issue

11. References

11.1. Normative References

[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.

[RFC6022] Scott, M. and M. Bjorklund, "YANG Module for NETCONF Monitoring", RFC 6022, October 2010.

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

11.2. Informative References

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

Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: April 10, 2015

D. Bogdanovic
Juniper Networks
K. Sreenivasa
Brocade Communications System
L. Huang
D. Blair
Cisco Systems

October 7, 2014

Network Access Control List (ACL) YANG Data Model
draft-bogdanovic-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 April 10, 2015.

Copyright Notice

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.

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

- 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 ususally 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 three 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, "packet-headers". The match container in "ietf-acl" uses groupings in "packet-headers". The "packet-headers" 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* [acl-name]
         +--rw acl-name               string
         +--rw acl-type?              acl-type
         +--ro acl-oper-data
         |  +--ro match-counter?   ietf:counter64
         |  +--ro targets*         string
         +--rw access-list-entries
            +--rw access-list-entry* [rule-name]
               +--rw rule-name        string
               +--rw matches
               |  +--rw (ace-type)?
               |  |  +--:(ace-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 ip-protocol?                     uint8
|  |  |  +--rw (ace-ip-version)?
|  |  |     +--:(ace-ipv4)
|  |  |     |  +--rw destination-ipv4-address?
                                                inet:ipv4-prefix
|  |  |     |  +--rw source-ipv4-address?
                                               inet:ipv4-prefix
|  |  |     +--:(ace-ipv6)
|  |  |        +--rw destination-ipv6-address?
                                                inet:ipv6-prefix
|  |  |        +--rw source-ipv6-address?
                                                inet:ipv6-prefix
|  |  |        +--rw flow-label?       inet:ipv6-flow-label
|  |  +--:(ace-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 ace-oper-data
   +--ro match-counter?   ietf:counter64
```

Module "newco-acl" is an example of company proprietary model, that augments "ietf-acl" module. It shows how 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 propietary models.

```
module: newco-acl
augment /ietf-acl:access-list/ietf-acl:access-list-entries/ietf-acl:matches:
   +--rw (protocol_payload_choice)?
      +--:(protocol_payload)
         +--rw protocol_payload* [value_keyword]
            +--rw value_keyword    enumeration
augment /ietf-acl:access-list/ietf-acl:access-list-entries/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
```

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 "packet-fields". The "actions" define behavior to undertake once a "match" has been identified.

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

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

  prefix acl;

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

  import 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).";

    revision 2014-10-10 {
      description "Creating base model for netmod.";
      reference
        "RFC 6020: YANG - A Data Modeling Language for the
        Network Configuration Protocol (NETCONF)";
    }

    identity acl-base {
      description "Base acl type for all ACL type identifiers.";
    }

    identity ip-acl {
      base "acl:acl-base";
      description "layer 3 ACL type";
    }
   identity eth-acl {
      base "acl:acl-base";
```

```
      description "layer 2 ACL type";
    }

    typedef acl-type {
      type identityref {
        base "acl-base";
      }
      description
        "This type is used to refer to an Access Control List
        (ACL) type";
    }

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

    container access-lists {
      description
        "Access control lists.";

      list access-list {
        key acl-name;
        description "
          An access list (acl) is an ordered list of
          access list entries (ace). Each ace has a
          sequence number to define the order, list
          of match criteria, and a list of actions.
          Since there are several kinds of acls
          implementeded with different attributes for
          each and different for each vendor, this
          model accomodates customizing acls for
          each kind and for each vendor.
          ";

        leaf acl-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 acl-type {
          type acl-type;
```

```
          description "Type of ACL";
        }

        container acl-oper-data {
          config false;

          description "Overall ACL operational data";
          leaf match-counter {
            type ietf:counter64;
            description "Total match count for ACL";
          }

          leaf-list targets {
            type string;
            description "List of targets where ACL 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 ace-type {
                  description "Type of ace.";
                  case ace-ip {
                    uses packet-fields:acl-ip-header-fields;
                    choice ace-ip-version {
                      description "Choice of IP version.";
                      case ace-ipv4 {
                        uses packet-fields:acl-ipv4-header-fields;
                      }
                      case ace-ipv6 {
                        uses packet-fields:acl-ipv6-header-fields;
                      }
                    }
                  }
```

```
                case ace-eth {
                  uses packet-fields:acl-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 ace-oper-data {
              config false;

              description "Per ace operational data";
              leaf match-counter {
                type ietf:counter64;
                description "Number of matches for an ace";
              }
            }
          }
        }
      }
    }
  }
```

4.2. Packet Header 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.

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

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

    prefix packet-fields;

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

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

    revision 2014-10-10 {
        description "Initial version of packet fields used by access-lists";
    }

    grouping acl-transport-header-fields {
        description "Transport header fields";

        container source-port-range {
             description "inclusive range of source ports";
             leaf lower-port {
                 mandatory true;
                 type inet:port-number;
             }
             leaf upper-port {
                 type inet:port-number;
             }
        }

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

```
    }

    grouping acl-ip-header-fields {
        description "Header fields common to ipv4 and ipv6";

        uses acl-transport-header-fields;

        leaf dscp {
            type inet:dscp;
        }

        leaf ip-protocol {
            type uint8;
        }

    }

    grouping acl-ipv4-header-fields {
        description "fields in IPv4 header";

        leaf destination-ipv4-address {
            type inet:ipv4-prefix;
        }

        leaf source-ipv4-address {
            type inet:ipv4-prefix;
        }

    }

    grouping acl-ipv6-header-fields {
        description "fields in IPv6 header";

        leaf destination-ipv6-address {
            type inet:ipv6-prefix;
        }

        leaf source-ipv6-address {
            type inet:ipv6-prefix;
        }

        leaf flow-label {
            type inet:ipv6-flow-label;
        }

    }

    grouping acl-eth-header-fields {
```

```
        description "fields in ethernet header";

        leaf destination-mac-address {
            type yang:mac-address;
        }

        leaf destination-mac-address-mask {
            type yang:mac-address;
        }

        leaf source-mac-address {
            type yang:mac-address;
        }

        leaf source-mac-address-mask {
            type yang:mac-address;
        }
    }

    grouping timerange {
        description "Define time range entries to restrict
            the access. The time range is identified by a name
            and then referenced by a function, so that those
            time restrictions are imposed on the function itself.";

        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 {
             description "Packet was received on this interface";
             type string;
        }
        uses timerange;
    }
}
```

4.3. A company proprietary module example

In the figure below is an example how proprietary models can be created on top of base ACL module. It is a simple example of how to use 'augment' with an XPath expression which extends instances of a particular type. In this example, all /ietf-acl:access-list/ietf-acl :access-list-entries/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-list/ietf-acl:access-list-entries/ ietf-acl:actions are augmented with new choice of actions. Here is an inclusive list of cases listed within a choice statement.

```
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 2014-05-21{
    description "creating newo 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 proprietry 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-entries/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";
          }
        }
      }
    }
}
```

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

4.4. An ACL Example

Requirement: Deny All traffic from 1.1.1.1 bound for host 2.2.2.2 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 1.1.1.1 host 2.2.2.2
```

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>
        <acl-name>sample-ip-acl</acl-name>
        <access-list-entries>
           <access-list-entry>
            <rule-name>telnet-block-rule</rule-name>
            <matches>
              <destination-ipv4-address>2.2.2.2/32</destination-ipv4-address>
              <source-ipv4-address>1.1.1.1/32</source-ipv4-address>
            </matches>
            <actions>
              <deny/>
            <actions/>
          </access-list-entry>
        </access-list-entries>
       </access-list>
     </access-lists>
   </top>
 </config>
   </edit-config>
 </rpc>
```

Figure 2

5. Extending existing model for route filtering

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. It is easy to extend the model below in the same way how the base ACL model can be extended with company proprietary extensions, described in the next section.

```
              module ietf-route-filter {
                yang-version 1;
```

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

                prefix ietf-route-filter;

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

                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 2014-08-15 {
                  description "creating Route-Filter extensions to ietf-acl mo
del";
                  reference " ";
```

```
                }

                augment "/ietf-acl:access-list/ietf-acl:access-list-entries/ie
tf-acl:matches"{
                  description "
                          This module augments the matches container in the ie
tf-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 rang
e";
                    leaf v4-lower-bound {
                      type ietf-types:ipv4-prefix;
                          description "Defines the lower IPv4 prefix/prefix le
ngth";
                    }
                    leaf v4-upper-bound {
                      type ietf-types: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 ietf-types:ipv6-prefix;
                      description "Defines the lower IPv6 prefix/prefix length
";
                    }
                    leaf v6-upper-bound {
                      type ietf-types:ipv6-prefix;
                      description "Defines the upper IPv6 prefix/prefix length
";
                    }
                  }
                    }
                  }
                }
              }
```

6. 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.

7. 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.

8. 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

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

9. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draf 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 proporitory ACL. The earlier draft was superseded with the new one that received more participation from many vendors.

10. Change log [RFC Editor: Please remove]

11. References

11.1. 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.

11.2. 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.

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
Intended status: Experimental
Expires: April 10, 2015

A. Clemm
J. Medved
E. Voit
Cisco Systems

October 7, 2014

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-02.txt

Abstract

This document introduces capabilities that allow YANG datastores to reference and incorporate information from remote datastores. This is accomplished by extending YANG with the ability to define mount points that act as references to data nodes in remote datastores, and by providing the necessary means to manage and administer those mount points. This facilitates the development of applications that need to access data that transcends individual network devices while improving network-wide object consistency.

This document also lays the groundwork for optional extensions to support subscriptions to remote object updates and transparent caching of objects. These options will speed application peformance without sacrificing data consistency.

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 April 10, 2015.

Table of Contents

1. Introduction

This document introduces a new capability that allows YANG datastores [RFC6020] to incorporate and reference information from remote datastores. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node as a "mount point", under which a remote datastore subtree can be mounted. To the user of the primary datastore, the remote information appears as an integral part of the datastore. It allows remote data nodes and datastore subtrees to be inserted into the local data hierarchy, arranged below local data nodes. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user’s machine.

The ability to mount information from remote datastores is new and not covered by existing YANG mechanisms. Until now, management information provided in a datastore has been intrinsically tied to the same server. In contrast, the capability introduced here allows the server to represent information from remote systems as if it were its own and contained in its own local data hierarchy.

YANG does provide means by which modules that have been separately defined can reference and augment one another. YANG also does provide means to specify data nodes that reference other data nodes. However, all the data is assumed to be instantiated as part of the same datastore, for example a datastore provided through a NETCONF server [RFC6241]. Existing YANG mechanisms do not account for the possibility that some information that needs to be referred not only resides in a different subtree of the same datastore, or was defined in a separate module that is also instantiated in the same datastore, but that is genuinely part of a different datastore that is provided by a different server.

The requirements for mounting YANG subtrees from remote datastores, as long as a set of associated use cases, are documented in [peermount-req]. The ability to mount data from remote datastores is useful to address various problems that several categories of applications are faced with:

One category of applications that can leverage this capability concerns network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface configuration data, which would be contained in a top-level container in a network element datastore, but may need to be contained in a list in a controller datastore in order to be able to distinguish instances from different network elements under the controller's scope. This in turn would require introduction of redundant YANG modules that effectively replicate the same information save for differences in hierarchy.

By directly mounting information from network element datastores, the controller does not need to replicate the same information from multiple datastores, nor does it need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions. Instead, the subtree of the remote system is attached to the local mount point. Operations that need to access data below the mount point are in effect transparently redirected to remote system, which is the authoritative owner of the data. The mounting system does not even necessarily need to be aware of the specific data in the remote subtree.

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters. When each network element maintains its own datastore with the same configurable settings, a single global change requires modifying the same information in many network elements across a network. In case of inconsistent configurations, network failures can result that are difficult to troubleshoot. In many cases, what is more desirable is the ability to configure such settings in a single place, then make them available to every network element. Today, this requires in general the introduction of specialized servers and configuration options outside the scope of NETCONF, such as RADIUS [RFC2866] or DHCP [RFC2131]. In order to address this within the scope of NETCONF and YANG, the same information would have to be redundantly modeled and maintained, representing operational data (mirroring some remote server) on some network elements and

configuration data on a designated master. Either way, additional complexity ensues.

Instead of replicating the same global parameters across different datastores, the solution presented in this document allows a single copy to be maintained in a subtree of single datastore that is then mounted by every network element that requires access to these parameters. The global parameters can be hosted in a controller or a designated network element. This considerably simplifies the management of such parameters that need to be known across elements in a network and require global consistency.

The capability of allowing to mount information from remote datastores into another datastore is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module and server needs to be aware of the concepts introduced here. Mounting is transparent to the models being mounted; any YANG model can be mounted.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

DHCP: Dynamic Host Configuration Protocol.

Datastore: A conceptual store of instantiated management 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 the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.

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.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

The following example scenarios outline some of the ways in which the ability to mount YANG datastores can be applied. Other mount topologies can be conceived in addition to the ones presented here.

3.1. Network controller view

Network controllers can use the mounting capability to present a consolidated view of management information across the network. This allows network controllers to expose network-wide abstractions, such as topologies or paths, multi-device abstractions, such as VRRP [RFC3768], and network-element specific abstractions, such as information about a network element's interfaces.

While an application on top of a controller could bypass the controller to access network elements directly for their element-specific abstractions, this would come at the expense of added inconvenience for the client application. In addition, it would compromise the ability to provide layered architectures in which access to the network by controller applications is truly channeled through the controller.

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element's, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data inconsistency and staleness. Equally importantly, it would lead to the redundant definition of data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network

elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

For example, "system"-level information of a network element would most naturally placed into a top-level container at that network element's datastore. At the same time, the same information in the context of the overall network, such as maintained by a controller, might better be provided in a list. For example, the controller might maintain a list with a list element for each network element, underneath which the network element's system-level information is contained. However, the containment structure of data nodes in a module, once defined, cannot be changed. This means that in the context of a network controller, a second module that repeats the same system-level information would need to be defined, implemented, and maintained. Any augmentations that add additional system-level information to the original module will likewise need to be redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore.

This scenario is depicted in Figure 1. In the figure, M1 is the mountpoint for the datastore in Network Element 1 and M2 is the mountpoint for the datastore in Network Element 2. MDN1 is the mounted data node in Network Element 1, and MDN2 is the mounted data node in Network Element 2.

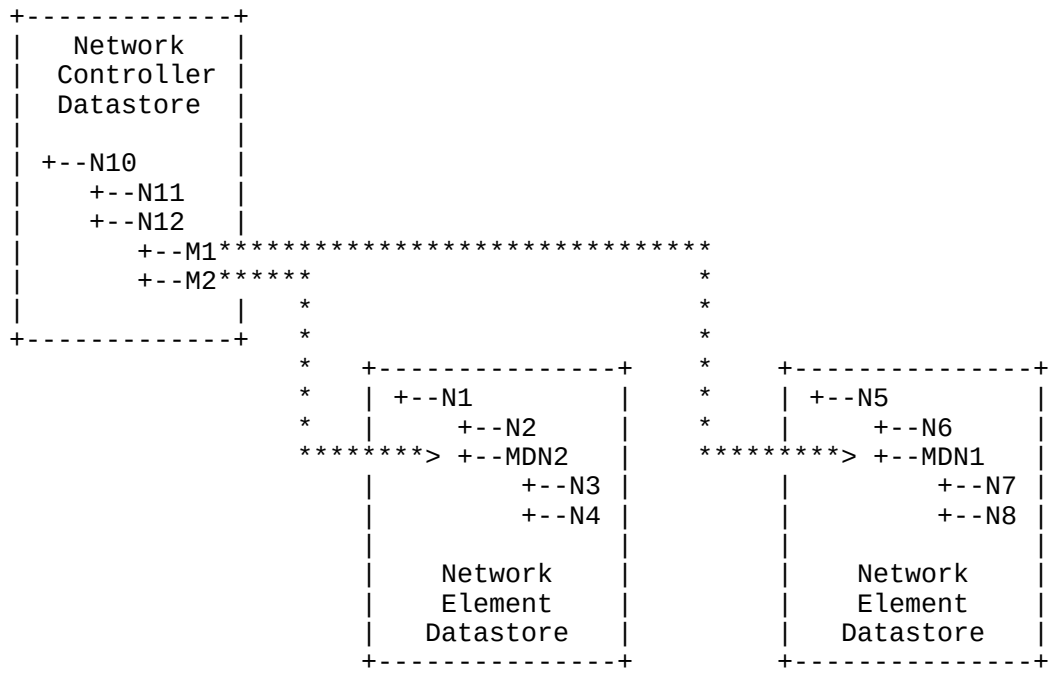


Figure 1: Network controller mount topology

3.2. Distributed network configuration

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters that need to be known across elements in a network. Today, the configuration of such parameters is generally performed on a per network element basis, which is not only redundant but, more importantly, error-prone. Inconsistent configurations lead to erroneous network behavior that can be challenging to troubleshoot.

Using the ability to mount information from remote datastores opens up a new possibility for managing such settings. Instead of replicating the same global parameters across different datastores, a single copy is maintained in a subtree of single datastore. This datastore can hosted in a controller or a designated network element. The subtree is subsequently mounted by every network element that requires access to these parameters.

In many ways, this category of applications is an inverse of the previous category: Whereas in the network controller case data from many different datastores would be mounted into the same datastore with multiple mountpoints, in this case many elements, each with

their own datastore, mount the same remote datastore, which is then mounted by many different systems.

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings.

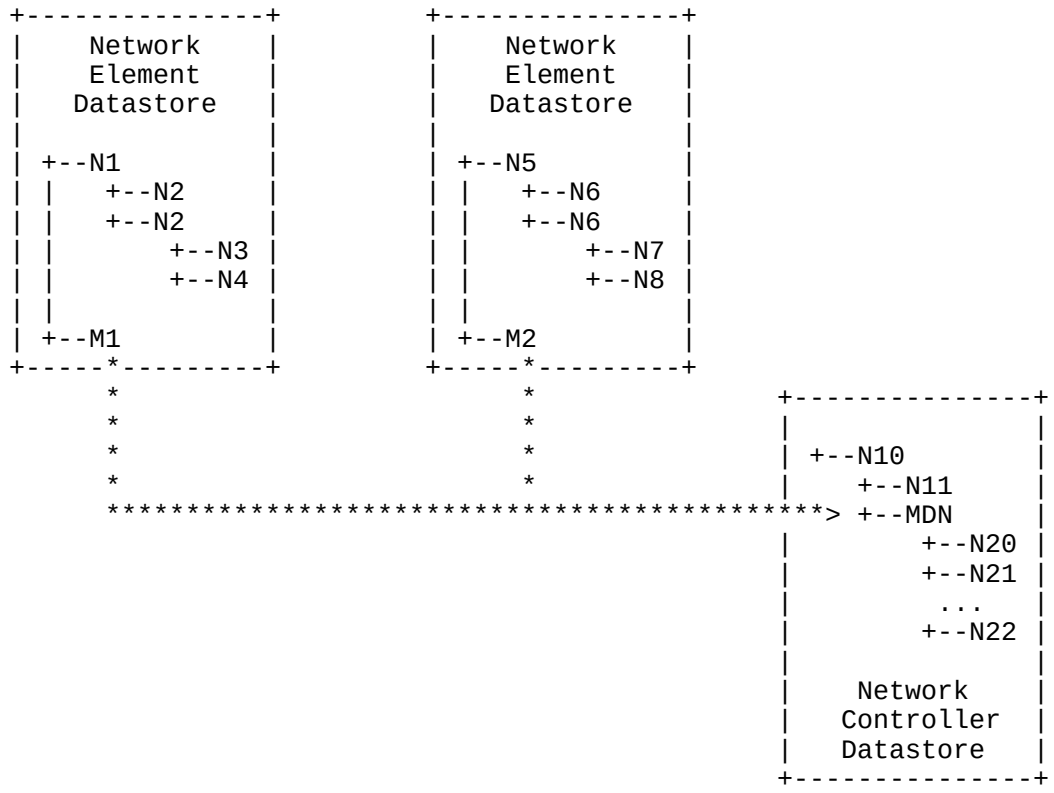


Figure 2: Distributed config settings topology

4. Operating on mounted data

This section provides a rough illustration of the operations flow involving mounted datastores.

4.1. General principles

The first thing that should be noted about these operations flows concerns the fact that a mount client essentially constitutes a special management application that interacts with a remote system. To the remote system, the mount client constitutes in effect just another application. The remote system is the authoritative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate the specific needs of the mount client to make it more efficient, the fact that another system decides to expose a certain "view" of that data is fundamentally not the remote system's concern.

When a client application makes a request to a server that involves data that is mounted from a remote system, the server will effectively act as a proxy to the remote system on the client application's behalf. It will extract from the client application request the portion that involves the mounted subtree from the remote system. It will strip that portion of the local context, i.e. remove any local data paths and insert the data path of the mounted remote subtree, as appropriate. The server will then forward the transposed request to the remote system that is the authoritative owner of the mounted data, acting itself as a client to the remote server. Upon receiving the reply, the server will transpose the results into the local context as needed, for example map the data paths into the local data tree structure, and combine those results with the results of the remainder portion of the original request.

4.2. Data retrieval

In the simplest and at the same time perhaps the most common case, the request will involve simple data retrieval. In that case, a "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system's subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

4.3. Data modification

Requests that involve editing of information and "writing through" to remote systems are potentially more complicated, particularly if transactions and locking across multiple configuration items are involved. However, these cases are not our primary concern at this

time. Data modifications that involve mounted information need to supported only in the following cases:

- When the scope of the operation falls within a single mountpoint. In that case, the data modification request (e.g. edit-config) results is directly passed through to the mount server. The mount client acts as a direct pass-through.

- When the modification involves no locking and no rollback, i.e. "best effort" semantics. In that case, the scope of the operation may extend beyond a single mountpoint.

This functionality is entirely sufficient for most use cases that need to be addressed. As outlined in [peermount-req], the aim for peer mount are use cases for which eventual consistency is sufficient and that do not require transactional consistency. As a result, the implementation is greatly simplified. Support for network-wide transactions and locking in conjunction with mount is not required. Servers MAY reject configuration requests involving commits and rollbacks, where the request involve datastore subtrees which include mount points below the root of the subtree. That said, it is conceivable to introduce in the future a special capability in which servers indicate that they provide such support.

By the same token, lock operations that extend across multiple datastores do not need to be supported. Lock requests on subtrees that include mount points MAY be rejected. That said, it is conceivable to introduce in the future a capability indicating that such a capability is supported. In order to perform a lock operation on a subtree that contains mount points, a server will need itself to obtain a lock from each of the respective remote mount servers before confirming the lock. If a lock cannot be obtained within a stringent timeout interval, the lock request will need to be denied and any locks that were already obtained released.

4.4. RPCs

YANG-Mount is aimed at data nodes in datastores. At this point, it does not extend towards RPCs that are defined as part of YANG modules whose contents is being mounted. Support for RPCs involving mounted portions of the datastore is for further study.

4.5. Notifications

YANG-Mount does not extend towards notifications. It is conceivable to offer such support in the future; however, at this point notification support involving mounted data nodes is for further study.

4.6. Other considerations

Since mounted information involves in general communication with a remote system, there is a possibility that the remote system does not respond within a certain amount of time, that connectivity is lost, or that other errors occur. Accordingly, the ability to mount datastores also involves mountpoint management, which includes the ability to configure timeouts, retries, and management of mountpoint state (including dynamic addition removal of mountpoints). Mountpoint management will be discussed in section Section 5.3.

It is expected that implementations will introduce caching schemes. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Caching is not required for YANG-mount to work - in which case all access to mounted information is "on-demand", in which the authoritative data node always gets accessed. Whether to perform caching is a local implementation decision. However, when caching is introduced, it can benefit from additional standardization, specifically the ability to subscribe to updates on remote data by remote servers. Some such optimizations to facilitate caching support will be discussed in section Section 5.4.

5. Data model structure

5.1. YANG mountpoint extensions

At the center of the module is a set of YANG extensions that allow to define a mountpoint.

- The first extension, "mountpoint", is used to declare a mountpoint. The extension takes the name of the mountpoint as an argument.

- The second extension, "target", serves as a substatement underneath a mountpoint statement. It takes an argument that identifies the target system. The argument is a reference to a data node that contains the information that is needed to identify and address a remote server, such as an IP address, a host name, or a URI [RFC3986].

- The third extension, "subtree", also serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression.

A mountpoint MUST be contained underneath a container. Future revisions might allow for mountpoints to be contained underneath other data nodes, such as lists, leaf-lists, and cases. However, to keep things simple, at this point mounting is only allowed directly underneath a container.

Only a single data node can be mounted at one time. While the mount target could refer to any data node, it is recommended that as a best practice, the mount target SHOULD refer to a container. It is possible to maintain e.g. a list of mount points, with each mount point each of which has a mount target an element of a remote list. However, to avoid unnecessary proliferation of the number of mount points and associated management overhead, when data from lists or leaf-lists is to be mounted, a container containing the list respectively leaf-list SHOULD be mounted instead of individual list elements.

It is possible for a mounted datastore to contain another mountpoint, thus leading to several levels of mount indirections. However, mountpoints MUST NOT introduce circular dependencies. In particular, a mounted datastore MUST NOT contain a mountpoint which specifies the mounting datastore as a target and a subtree which contains as root node a data node that in turn contains the original mountpoint. Whenever a mount operation is performed, this condition mountpoint. Whenever a mount operation is performed, this condition MUST be validated by the mount client.

5.2. YANG structure diagrams

YANG data model structure overviews have proven very useful to convey the "Big Picture". It would be useful to indicate in YANG data model structure overviews the fact that a given data node serves as a mountpoint. We propose for this purpose also a corresponding extension to the structure representation convention. Specifically, we propose to prefix the name of the mounting data node with upper-case ’M’.

```
rw network
+-- rw nodes
    +-- rw node [node-ID]
        +-- rw node-ID
        +-- M node-system-info
```

5.3. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

For this purpose, a list of the mountpoints is introduced. Each list element represents a single mountpoint. It includes an identification of the mount target, i.e. the remote system hosting the remote datastore and a definition of the subtree of the remote data node being mounted. It also includes monitoring information about current status (indicating whether the mount has been successful and is operational, or whether an error condition applies such as the target being unreachable or referring to an invalid subtree).

In addition to the list of mountpoints, a set of global mount policy settings allows to set parameters such as mount retries and timeouts.

Each mountpoint list element also contains a set of the same configuration knobs, allowing administrators to override global mount policies and configure mount policies on a per-mountpoint basis if needed.

There are two ways how mounting occurs: automatic (dynamically performed as part of system operation) or manually (administered by a user or client application). A separate mountpoint-origin object is used to distinguish between manually configured and automatically populated mountpoints.

Whether mounting occurs automatically or needs to be manually configured by a user or an application can depend on the mountpoint being defined, i.e. the semantics of the model.

When configured automatically, mountpoint information is automatically populated by the datastore that implements the mountpoint. The precise mechanisms for discovering mount targets and bootstrapping mount points are provided by the mount client infrastructure and outside the scope of this specification. Likewise, when a mountpoint should be deleted and when it should merely have its mount-status indicate that the target is unreachable is a system-specific implementation decision.

Manual mounting consists of two steps. In a first step, a mountpoint is manually configured by a user or client application through administrative action. Once a mountpoint has been configured, actual mounting occurs through an RPCs that is defined specifically for that purpose. To unmount, a separate RPC is invoked; mountpoint configuration information needs to be explicitly deleted. Manual mounting can also be used to override automatic mounting, for example to allow an administrator to set up or remove a mountpoint.

It should be noted that mountpoint management does not allow users to manually "extend" the model, i.e. simply add a subtree underneath

some arbitrary data node into a datastore, without a supporting mountpoint defined in the model to support it. A mountpoint definition is a formal part of the model with well-defined semantics. Accordingly, mountpoint management does not allow users to dynamically "extend" the data model itself. It allows users to populate the datastore and mount structure within the confines of a model that has been defined prior.

The structure of the mountpoint management data model is depicted in the following figure, where brackets enclose list keys, "rw" means configuration, "ro" operational state data, and "?" designates optional nodes. Parantheses enclose choice and case nodes. The figure does not depict all definitions; it is intended to illustrate the overall structure.

```
rw mount-server-mgmt
+-- rw mountpoints
|   +-- rw mountpoint [mountpoint-id]
|       +-- rw mountpoint-id  string
|       +-- rw mount-target
|       |   +--: (IP)
|       |   |    +-- rw target-ip  yang:ip-address
|       |   +--: (URI)
|       |   |    +-- rw uri  yang:uri
|       |   +--: (host-name)
|       |   |    +-- rw hostname  yang:host
|       |   +-- (node-ID)
|       |   |    +-- rw node-info-ref  mnt:subtree-ref
|       |   +-- (other)
|       |        +-- rw opaque-target-id  string
|       +-- rw subtree-ref  mnt:subtree-ref
|       +-- ro mountpoint-origin enumeration
|       +-- ro mount-status  mnt:mount-status
|       +-- rw manual-mount? empty
|       +-- rw retry-timer? uint16
|       +-- rw number-of-retries? uint8
+-- rw global-mount-policies
    +-- rw manual-mount? empty
    +-- rw retry-time? uint16
    +-- rw number-of-retries? uint8
```

5.4. Caching

Under certain circumstances, it can be useful to maintain a cache of remote information. Instead of accessing the remote system, requests are served from a copy that is locally maintained. This is particularly advantageous in cases where data is slow changing, i.e. when there are many more "read" operations than changes to the

underlying data node, and in cases when a significant delay were incurred when accessing the remote system, which might be prohibitive for certain applications. Examples of such applications are applications that involve real-time control loops requiring response times that are measured in milliseconds.

Caching can in principle apply to both retrieval and modification operations. However, as data nodes that are mounted from an authoritative datastore represent the "golden copy", it is important that any modifications are reflected as soon as they are made. Likewise, typical applications that operate on YANG datastores will not apply high frequency changes to the same data nodes. For those reasons, the focus in the following is on caching for data retrieval purposes. Caching for operations that involve change operations are in the following not considered.

It is a local implementation decision of mount clients whether to cache information once it has been fetched. However, in order to support more powerful caching schemes, it becomes necessary for the mount server to "push" information proactively. This means that at this point, the mount server is no longer oblivious to the fact that a mount client exists.

For this purpose, we are planning in a subsequent revision to introduce caching extensions. The following outlines what these extensions will entail.

The first set of extensions concern the mount client. We are adding an extension to mountpoint management that allows a mount client to define a specific binding type for a given mount point. A mount binding specifies how the client wishes to have information from a remote system populated. The following binding types are defined:

- On-demand. This is the "default" binding. No caching is applied. Information is always retrieved from the remote datastore, whenever a client application requests it.

- Periodic. In this case, the mounted data is updated periodically. The interval in which updates are to take place can be parametrized.

- On-change. In this case, mounted data is updated whenever a change is detected. In order to reduce the risk of churn in the case of fast-changing data, a dampening interval can be specified, indicating the minimum time that must pass between updates. Further extensions can allow to specify the magnitude or size a change must indicate in order to be reported.

The second set of extensions concern the mount server. NETCONF and RESTconf are fundamentally request-response based protocols. In order to support periodic and, even more so, on-change binding types, it is advantageous if the remote server supports a mechanism that allows a mount client to subscribe to data in a datastore subtree and then have that data be automatically delivered without requiring further requests. Certainly, resorting to polling should be avoided! There are different mechanisms conceivable for this, such as the support of information push or publish/subscribe.

Data subscription mechanisms can be of interest beyond YANG-Mount. However, at this point, such a mechanism has not yet been defined. The following outlines one way in which this can be achieved.

One way in which this can be achieved is through simple NETCONF notifications and a special data subscription function, whose configuration can be expressed through YANG itself.

The notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.
- A data node that contains a representation of the datastore subtree. (This can be simply a node of type string or, for XML-based encoding, anyxml.)

The configuration of the subscription in turn contains several parameters as well:

- The root of the data subtree being subscribed to
- The identity of the subscriber(s)
- The subscription type: periodic or on change
- For periodic subscriptions: the start time and interval with which to push updates
- For change-based subscriptions: the dampening interval with which to push repeated changes, an indicator for the magnitude of changes, etc

5.5. Other considerations

5.5.1. Authorization

Whether a mount client is allowed to modify information in a mounted datastore or only retrieve it and whether there are certain data nodes or subtrees within the mounted information for which access is restricted is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

Users and implementers need to be aware of certain issues when mounted information is modified, not just retrieved. Specifically, in certain corner cases validation of changes made to mounted data may involve constraints that involve information that is not visible to the mounting datastore. This means that in such cases the reason for validation failures may not always be fully understood by the mounting system.

Likewise, if the concepts of transactions and locking are applied at the mounting system, these concepts will need to be applied across multiple systems, not just across multiple data nodes within the same system. This capability may not be supported by every implementation. For example, locking a datastore that contains a mountpoint requires that the mount client obtains corresponding locks on the mounted datastore as needed. Any request to acquire a lock on a configuration subtree that includes a mountpoint MUST NOT be granted if the mount client fails to obtain a corresponding lock on the mounted system. Likewise, in case transactions are supported by the mounting system, but not the target system, requests to acquire a lock on a configuration subtree that includes a mountpoint MUST NOT be granted.

5.5.2. 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". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

It is conceivable to use mount in conjunction with ephemeral datastores, to address requirements outlined in [draft-haas-i2rs-netmod-netconf-requirements]. Support for such a

scheme is for further study and may be included in a future revision of this spec.

5.5.3. 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.

5.5.4. 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.

5.5.5. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, the following considerations apply:

Systems that wish to mount information from remote datastores need to implement a mount client. The mount client communicates with a remote system to access the remote datastore. To do so, there are several options:

- The mount client acts as a NETCONF client to a remote system. Alternatively, another interface to the remote system can be used, such as a REST API using JSON encodings, as specified in [I-D.ietf-netconf-restconf]. Either way, to the remote system, the mount client constitutes essentially a client application like any other. The mount client in effect IS a special kind of client application.

- The mount client communicates with a remote mount server through a separate protocol. The mount server is deployed on the same system as the remote NETCONF datastore and interacts with it through a set of local APIs.

- The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client’s behalf. The communication between mount client and remote mount

server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

It is the responsibility of the mount client to manage the association with the target system, e.g. validate it is still reachable by maintaining a permanent association, perform reachability checks in case of a connectionless transport, etc.

It is the responsibility of the mount client to manage the mountpoints. This means that the mount client needs to populate the mountpoint monitoring information (e.g. keep mount-status up to data and determine in the case of automatic mounting when to add and remove mountpoint configuration). In the case of automatic mounting, the mount client also interacts with the mountpoint discovery and bootstrap process.

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested involving a mountpoint is relayed by the mounting system's infrastructure to the mount client. For example, a request to retrieve information from a datastore leads to an invocation of an internal mount client API when a mount point is reached. The mount client then relays a corresponding operation to the remote datastore. It subsequently relays the result along with any responses back to the invoking infrastructure, which then merges the result (e.g. a retrieved subtree with the rest of the information that was retrieved) as needed. Relaying the result may involve the need to transpose error response codes in certain corner cases, e.g. when mounted information could not be reached due to loss of connectivity with the remote server, or when a configuration request failed due to validation error.

5.5.6. Modeling best practices

There is a certain amount of overhead associated with each mount point. The mount point needs to be managed and state maintained. Data subscriptions need to be maintained. Requests including mounted subtrees need to be decomposed and responses from multiple systems combined.

For those reasons, as a general best practice, models that make use of mount points SHOULD be defined in a way that minimizes the number of mountpoints required. Finely granular mounts, in which multiple mountpoints are maintained with the same remote system, each containing only very small data subtrees, SHOULD be avoided. For example, lists SHOULD only contain mountpoints when individual list elements are associated with different remote systems. To mount data from lists in remote datastores, a container node that contains all

list elements SHOULD be mounted instead of mounting each list element individually. Likewise, instead of having mount points refer to nodes contained underneath choices, a mountpoint should refer to a container of the choice.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "mount@2014-10-07.yang"
module mount {
    namespace "urn:cisco:params:xml:ns:yang:mount";
    // replace with IANA namespace when assigned

    prefix mnt;

    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: Alexander Clemm
        alex@cisco.com";

    description
        "This module provides a set of YANG extensions and definitions
         that can be used to mount information from remote datastores.";

    revision 2014-10-07 {
        description "Initial revision.";
    }

    feature mount-server-mgmt {
        description
            "Provide additional capabilities to manage remote mount
             points";
    }
```

```
    extension mountpoint {
        description
            "This YANG extension is used to mount data from a remote
            system in place of the node under which this YANG extension
            statement is used.

            This extension takes one argument which specifies the name
            of the mountpoint.

            This extension can occur as a substatement underneath a
            container statement, a list statement, or a case statement.
            As a best practice, it SHOULD occur as statement only
            underneath a container statement, but it MAY also occur
            underneath a list or a case statement.

            The extension takes two parameters, target and subtree, each
            defined as their own YANG extensions.
            A mountpoint statement MUST contain a target and a subtree
            substatement for the mountpoint definition to be valid.

            The target system MAY be specified in terms of a data node
            that uses the grouping 'mnt:mount-target'.  However, it
            can be specified also in terms of any other data node that
            contains sufficient information to address the mount target,
            such as an IP address, a host name, or a URI.

            The subtree SHOULD be specified in terms of a data node of
            type 'mnt:subtree-ref'. The targeted data node MUST
            represent a container.

            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.";

        argument "name";
    }

    extension target {
        description
            "This YANG extension is used to specify a remote target
             system from which to mount a datastore subtree.  This YANG
             extension takes one argument which specifies the remote
             system. In general, this argument will contain the name of
             a data node that contains the remote system information. It
             is recommended that the reference data node uses the
```

```
            mount-target grouping that is defined further below in this
            module.

            This YANG extension can occur only as a substatement below
            a mountpoint statement. It MUST NOT occur as a substatement
            below any other YANG statement.";

        argument "target-name";
    }

    extension subtree {
        description
            "This YANG extension is used to specify a subtree in a
            datastore that is to be mounted.  This YANG extension takes
            one argument which specifies the path to the root of the
            subtree. The root of the subtree SHOULD represent an
            instance of a YANG container.  However, it MAY represent
            also another data node.

            This YANG extension can occur only as a substatement below
            a mountpoint statement. It MUST NOT occur as a substatement
            below any other YANG statement.";

        argument "subtree-path";
    }

    typedef mount-status {
        description
            "This type is used to represent the status of a
            mountpoint.";
        type enumeration {
            enum ok; {
                description
                    "Mounted";
            }
            enum no-target {
                description
                    "The argument of the mountpoint does not define a
                     target system";
            }
            enum no-subtree {
                description
                    "The argument of the mountpoint does not define a
                      root of a subtree";
            }
            enum target-unreachable {
                description
                    "The specified target system is currently
```

```
                    unreachable";
            }
            enum mount-failure {
                description
                    "Any other mount failure";
            }
            enum unmounted {
                description
                    "The specified mountpoint has been unmounted as the
                    result of a management operation";
            }
        }
    }
    typedef subtree-ref {
        type string;  // string pattern to be defined
        description
            "This string specifies a path to a datanode. It corresponds
            to the path substatement of a leafref type statement.  Its
            syntax needs to conform to the corresponding subset of the
            XPath abbreviated syntax. Contrary to a leafref type,
            subtree-ref allows to refer to a node in a remote datastore.
            Also, a subtree-ref refers only to a single node, not a list
            of nodes.";
    }
    rpc mount {
        description
            "This RPC allows an application or administrative user to
            perform a mount operation.  If successful, it will result in
            the creation of a new mountpoint.";
        input {
            leaf mountpoint-id {
                type string {
                    length "1..32";
                }
            }
        }
        output {
            leaf mount-status {
                type mount-status;
            }
        }
    }
    rpc unmount {
        "This RPC allows an application or administrative user to
        unmount information from a remote datastore.  If successful,
        the corresponding mountpoint will be removed from the
        datastore.";
        input {
```

```
            leaf mountpoint-id {
                type string {
                    length "1..32";
                }
            }
        }
        output {
            leaf mount-status {
                type mount-status;
            }
        }
    }
    grouping mount-monitor {
        leaf mount-status {
            description
                "Indicates whether a mountpoint has been successfully
                mounted or whether some kind of fault condition is
                present.";
            type mount-status;
            config false;
        }
    }
    grouping mount-target {
        description
            "This grouping contains data nodes that can be used to
            identify a remote system from which to mount a datastore
            subtree.";
        container mount-target {
            choice target-address-type {
                mandatory;
                case IP {
                    leaf target-ip {
                        type yang:ip-address;
                    }
                case URI {
                    leaf uri {
                        type yang:uri;
                    }
                }
                case host-name {
                    leaf hostname {
                        type yang:host;
                    }
                }
                case node-ID {
                    leaf node-info-ref {
                        type subtree-ref;
                    }
```

```
                }
                case other {
                    leaf opaque-target-ID {
                        type string;
                        description
                            "Catch-all; could be used also for mounting
                            of data nodes that are local.";
                    }
                }
            }
        }
    }
    grouping mount-policies {
        description
            "This grouping contains data nodes that allow to configure
            policies associated with mountpoints.";
        leaf manual-mount {
            type empty;
            description
                "When present, a specified mountpoint is not
                automatically mounted when the mount data node is
                created, but needs to mounted via specific RPC
                invocation.";
        }
        leaf retry-timer {
            type uint16;
            units "seconds";
            description
                "When specified, provides the period after which
                mounting will be automatically reattempted in case of a
                mount status of an unreachable target";
        }
        leaf number-of-retries {
            type uint8;
            description
                "When specified, provides a limit for the number of
                times for which retries will be automatically
                attempted";
        }
    }

    container mount-server-mgmt {
        if-feature mount-server-mgmt;
        container mountpoints {
            list mountpoint {
                key "mountpoint-id";

                leaf mountpoint-id {
```

```
                    type string {
                        length "1..32";
                    }
                }
                leaf mountpoint-origin {
                    type enumeration {
                        enum client {
                            description
                                "Mountpoint has been supplied and is
                                manually administered by a client";
                        }
                        enum auto {
                            description
                                "Mountpoint is automatically
                                 administered by the server";
                        }
                    config false;
                    }
                }
                uses mount-target;
                leaf subtree-ref {
                    type subtree-ref;
                    mandatory;
                }
                uses mount-monitor;
                uses mount-policies;
            }
        }
        container global-mount-policies {
            uses mount-policies;
            description
                "Provides mount policies applicable for all mountpoints,
                unless overridden for a specific mountpoint.";
        }
    }
}
<CODE ENDS>
```

7. Security Considerations

TBD

8. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Tony Tkacik, Ambika Tripathy, Robert Varga, Prabhakara Yellai, Shashi Kumar Bansal, Lukas Sedlak, and Benoit Claise.

9. References

9.1. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC3768] Hinden, R., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, April 2004.

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

[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.

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

9.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-ietf-netconf-restconf-01 (work in progress), July 2014.

[draft-haas-i2rs-netmod-netconf-requirements]
Haas, J., "I2RS Requirements for Netmod/Netconf", draft-haas-i2rs-netmod-netconf-requirements-02 (work in progress), September 2014.

[peermount-req]
Voit, E., Clemm, A., Bansal, S., Tripathy, A., and P. Yellai, "Requirements for Peer Mounting of YANG subtrees from Remote Datastores", draft-voit-netmod-peer-mount-requirements-00 (work in progress), September 2014.

Appendix A. Example

In the following example, we are assuming the use case of a network controller that wants to provide a controller network view to its client applications. This view needs to include network abstractions that are maintained by the controller itself, as well as certain information about network devices where the network abstractions tie in with element-specific information. For this purpose, the network controller leverages the mount capability specified in this document and presents a fictitious Controller Network YANG Module that is depicted in the outlined structure below. The example illustrates how mounted information is leveraged by the mounting datastore to provide an additional level of information that ties together network and device abstractions, which could not be provided otherwise without introducing a (redundant) model to replicate those device abstractions

```
rw controller-network
+-- rw topologies
|   +-- rw topology [topo-id]
|       +-- rw topo-id                 node-id
|       +-- rw nodes
|       |   +-- rw node [node-id]
|       |       +-- rw node-id         node-id
|       |       +-- rw supporting-ne   network-element-ref
|       |       +-- rw termination-points
|       |           +-- rw term-point [tp-id]
|       |               +-- tp-id      tp-id
|       |               +-- ifref      mountedIfRef
|       +-- rw links
|           +-- rw link [link-id]
|               +-- rw link-id         link-id
|               +-- rw source          tp-ref
|               +-- rw dest            tp-ref
+-- rw network-elements
    +-- rw network-element [element-id]
        +-- rw element-id              element-id
        +-- rw element-address
        |   +-- ...
        +-- M interfaces
```

The controller network model consists of the following key components:

- A container with a list of topologies. A topology is a graph representation of a network at a particular layer, for example, an IS-IS topology, an overlay topology, or an Openflow topology. Specific topology types can be defined in their own separate YANG

modules that augment the controller network model. Those augmentations are outside the scope of this example

- An inventory of network elements, along with certain information that is mounted from each element. The information that is mounted in this case concerns interface configuration information. For this purpose, each list element that represents a network element contains a corresponding mountpoint. The mountpoint uses as its target the network element address information provided in the same list element

- Each topology in turn contains a container with a list of nodes. A node is a network abstraction of a network device in the topology. A node is hosted on a network element, as indicated by a network-element leafref. This way, the "logical" and "physical" aspects of a node in the network are cleanly separated.

- A node also contains a list of termination points that terminate links. A termination point is implemented on an interface. Therefore, it contains a leafref that references the corresponding interface configuration which is part of the mounted information of a network element. Again, the distinction between termination points and interfaces provides a clean separation between logical concepts at the network topology level and device-specific concepts that are instantiated at the level of a network element. Because the interface information is mounted from a different datastore and therefore occurs at a different level of the containment hierarchy than it would if it were not mounted, it is not possible to use the interface-ref type that is defined in YANG data model for interface management [] to allow the termination point refer to its supporting interface. For this reason, a new type definition "mountedIfRef" is introduced that allows to refer to interface information that is mounted and hence has a different path.

- Finally, a topology also contains a container with a list of links. A link is a network abstraction that connects nodes via node termination points. In the example, directional point-to-point links are depicted in which one node termination point serves as source, another as destination.

The following is a YANG snippet of the module definition which makes use of the mountpoint definition.

```
<CODE BEGINS>
module controller-network {
    namespace "urn:cisco:params:xml:ns:yang:controller-network";
    // example only, replace with IANA namespace when assigned
    prefix cn;
    import mount {
        prefix mnt;
    }
    import interfaces {
        prefix if;
    }
    ...
    typedef mountedIfRef {
        type leafref {
            path "/cn:controller-network/cn:network-elements/"
            +"cn:network-element/cn:interfaces/if:interface/if:name";
            //  cn:interfaces corresponds to the mountpoint
        }
    }
    ...
    list termination-point {
        key "tp-id";
        ...
        leaf ifref {
            type mountedIfRef;
        }
        ...
        list network-element {
            key "element-id";
            leaf element-id {
                type element-ID;
            }
            container element-address {
                ... // choice definition that allows to specify
                // host name,
                // IP addresses, URIs, etc
            }
            mnt:mountpoint "interfaces" {
                mnt:target "./element-address";
                mnt:subtree "/if:interfaces";
            }
        ...
    }
...
<CODE ENDS>
```

Finally, the following contains an XML snippet of instantiated YANG information. We assume three datastores: NE1 and NE2 each have a

datastore (the mount targets) that contains interface configuration data, which is mounted into NC's datastore (the mount client).

Interface information from NE1 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/1</name>
    <name>ethernetCsmacd</type>
    <location>1/1</location>
  </interface>
<interfaces>
```

Interface information from NE2 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/2</name>
    <name>ethernetCsmacd</type>
    <location>1/2</location>
  </interface>
<interfaces>
```

NC datastore with mounted interface information from NE1 and NE2:

```
   <controller-network>
     ...
     <network-elements>
       <network-element>
         <element-id>NE1</element-id>
         <element-address> .... </element-address>
         <interfaces>
           <if:interface>
             <if:name>fastethernet-1/0</if:name>
             <if:type>ethernetCsmacd</if:type>
             <if:location>1/0</if:location>
           </if:interface>
           <if:interface>
             <if:name>fastethernet-1/1</if:name>
             <if:type>ethernetCsmacd</if:type>
             <if:location>1/1</if:location>
           </if:interface>
         <interfaces>
       </network-element>
       <network-element>
         <element-id>NE2</element-id>
         <element-address> .... </element-address>
         <interfaces>
           <if:interface>
             <if:name>fastethernet-1/0</if:name>
             <if:type>ethernetCsmacd</if:type>
             <if:location>1/0</if:location>
           </if:interface>
           <if:interface>
             <if:name>fastethernet-1/2</if:name>
             <if:type>ethernetCsmacd</if:type>
             <if:location>1/2</if:location>
           </if:interface>
         <interfaces>
       </network-element>
     </network-elements>
     ...
   </controller-network>
```

Authors' Addresses

Alexander Clemm
Cisco Systems

EMail: alex@cisco.com

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Network Working Group
Internet-Draft
Obsoletes: 6020 (if approved)
Intended status: Standards Track
Expires: April 5, 2015

M. Bjorklund, Ed.
Tail-f Systems

October 2, 2014

YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)

draft-ietf-netmod-rfc6020bis-01

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

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 April 5, 2015.

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].

- Changed the YANG version from "1" to "1.1".
- Made the "yang-version" statement mandatory.
- 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.
- Extended the "if-feature" syntax to be a boolean expression over feature names.
- Added a set of new XPath functions in Section 10.
- Added a new substatement "modifier" to pattern (see Section 9.4.6).

- Defined the string value of an identityref in XPath expressions (see Section 9.10).
- Allow "if-feature" in "refine".
- Made "when" and "if-feature" illegal on list keys, unless the parent is also conditional, and the condition matches the parent's condition.
- Allow "choice" as a shorthand case statement (see Section 7.9).

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

- anyxml: A data node that can contain an unknown chunk of XML data.
- augment: Adds new schema nodes to a previously defined schema node.
- base type: The type from which a derived type was derived, which may be either a built-in type or another derived type.
- built-in type: A YANG data type defined in the YANG language, such as uint32 or string.
- choice: A schema node where only one of a number of identified alternatives is valid.
- configuration data: The set of writable data that is required to transform a system from its initial default state into its current state [RFC6241].
- conformance: A measure of how accurately a device follows a data model.
- 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.
- data definition statement: A statement that defines new data nodes. One of container, leaf, leaf-list, list, choice, case, augment, uses, and anyxml.

- data model: A data model describes how data is represented and accessed.

- 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.

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

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

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

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

- 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.

- 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.

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

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

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

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

- 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.

- 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.

- 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".

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

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

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

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

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

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

- 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.

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

- 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:

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

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

- 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 "include" statement allows a module or submodule to reference material in submodules, and the "import" statement allows references to material defined in other modules.

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 is a sequence of leaf nodes with exactly one value of a particular type per leaf.

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, 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 7.15.

4.2.9. RPC Definitions

YANG allows the definition of NETCONF RPCs. The operations' names, input parameters, and output parameters are modeled using YANG data definition statements.

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.

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 7.14.

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 its submodules, and the "import" statement to reference external modules. Similarly, a submodule uses the "import" statement to reference other modules, and uses the "include" statement to reference other submodules within its 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 to other modules and submodules:

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

- For a module to reference definitions in one of its submodules, the module MUST include the submodule.

- For a submodule to reference definitions in a second submodule of the same module, the first submodule MUST include the second submodule.

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

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 and <error-info> content. 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 7.18.1.

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 7.18.3.

5.6.4. Announcing Conformance Information in the <hello> Message

The namespace URI MUST be advertised as a capability in the NETCONF <hello> message to indicate support for the YANG module by a NETCONF server. The capability URI advertised MUST be of the form:

```
  capability-string   = namespace-uri [ parameter-list ]
  parameter-list      = "?" parameter *( "&" parameter )
  parameter           = revision-parameter /
                        module-parameter /
                        feature-parameter /
                        deviation-parameter
  revision-parameter  = "revision=" revision-date
  module-parameter    = "module=" module-name
  feature-parameter   = "features=" feature *( "," feature )
  deviation-parameter = "deviations=" deviation *( "," deviation )
```

Where "revision-date" is the revision of the module (see Section 7.1.9) that the NETCONF server implements, "module-name" is the name of module as it appears in the "module" statement (see Section 7.1), "namespace-uri" is the namespace URI for the module as it appears in the "namespace" statement (see Section 7.1.3), "feature" is the name of an optional feature implemented by the device (see Section 7.18.1), and "deviation" is the name of a module defining device deviations (see Section 7.18.3).

In the parameter list, each named parameter MUST occur at most once.

5.6.4.1. Modules

Servers indicate the names of supported modules via the <hello> message. Module namespaces are encoded as the base URI in the capability string, and the module name is encoded as the "module" parameter to the base URI.

A server MUST advertise all revisions of all modules it implements.

For example, this <hello> message advertises one module "syslog".

```
  <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
      <!-- wrapped for display only -->
      <capability>
        http://example.com/syslog?module=syslog&amp;
        revision=2008-04-01
      </capability>
    </capabilities>
    <session-id>42</session-id>
  </hello>
```

5.6.4.2. Features

Servers indicate the names of supported features via the <hello> message. In <hello> messages, the features are encoded in the "features" parameter within the URI. The value of this parameter is a comma-separated list of feature names that the device supports for the specific module.

For example, this <hello> message advertises one module, informing the client that it supports the "local-storage" feature of module "syslog".

```
  <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
      <!-- wrapped for display only -->
      <capability>
        http://example.com/syslog?module=syslog&amp;
        features=local-storage
      </capability>
    </capabilities>
    <session-id>43</session-id>
  </hello>
```

5.6.4.3. Deviations

Device deviations are announced via the "deviations" parameter. The value of the "deviations" parameter is a comma-separated list of modules containing deviations from the capability's module.

For example, this <hello> message advertises two modules, informing the client that it deviates from module "syslog" according to the deviations listed in the module "my-devs".

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <!-- wrapped for display only -->
    <capability>
      http://example.com/syslog?module=syslog&amp;
      deviations=my-devs
    </capability>
    <capability>
      http://example.com/my-deviations?module=my-devs
    </capability>
  </capabilities>
  <session-id>44</session-id>
</hello>
```

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.

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.

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

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

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

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

- 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.

- 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.

- 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.

- 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 7.17). 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.

Since submodules cannot include the parent module, any extensions in the module that need to be exposed to submodules MUST be defined in a submodule. Submodules can then include this submodule to find the definition of the extension.

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.

6.4.1. XPath Context

All YANG XPath expressions share the following XPath context definition:

- 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.

- 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).

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

- 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 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	7.15	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.19.3	0..1
deviation	7.18.3	0..n
extension	7.17	0..n
feature	7.18.1	0..n
grouping	7.11	0..n
identity	7.16	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	7.14	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.19.4	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:

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

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

- 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, or to another submodule of that 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. When a submodule includes another submodule, the target submodule's definitions are made available to the current submodule.

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	7.19.3	0..1
reference	7.19.4	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>
        <include 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	7.15	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	7.19.3	0..1
deviation	7.18.3	0..n
extension	7.17	0..n
feature	7.18.1	0..n
grouping	7.11	0..n
identity	7.16	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	7.14	0..n
organization	7.1.7	0..1
reference	7.19.4	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 local submodule and any included 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, in modules or submodules which include it, 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	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	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	7.16.2	0..1
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
anyxml	7.10	0..n
choice	7.9	0..n
config	7.19.1	0..1
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	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	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
uses	7.12	0..n
when	7.19.5	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 data tree, and for all leafs with default values in use (see Section 7.6.1). If a data node does not exist in the data tree, and it does not have a default value, its "must" statements are not evaluated.

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:

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

- The accessible tree is made up of all nodes in the data tree, and all leafs with default values in use (see Section 7.6.1).

The accessible tree depends on the context node:

- If the context node represents configuration, the tree is the data in the NETCONF datastore where the context node exists. The XPath root node has all top-level configuration data nodes in all modules as children.

- If the context node represents state data, the tree is all state data on the device, and the "running" datastore. The XPath root node has all top-level data nodes in all modules as children.

- If the context node represents notification content, the tree is the notification XML instance document. The XPath root node has the element representing the notification being defined as the only child.

- If the context node represents RPC input parameters, the tree is the RPC XML instance document. The XPath root node has the element representing the RPC operation being defined as the only child.

- If the context node represents RPC output parameters, the tree is the RPC reply instance document. The XPath root node has the elements representing the RPC output parameters as children.

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	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	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:

- If no such ancestor exists in the schema tree, the default value MUST be used.
- 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.
- 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	7.19.1	0..1
default	7.6.4	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.19.5	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):

- If no such ancestor exists in the schema tree, the leaf MUST exist.

- Otherwise, if this ancestor is a case node, the leaf MUST exist if any node from the case exists in the data tree.

- 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).

If the type referenced by the leaf-list has a default value, it has no effect in the leaf-list.

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 7.7.5.

7.7.2. The leaf-list's Substatements

substatement	section	cardinality
config	7.19.1	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
max-elements	7.7.4	0..1
min-elements	7.7.3	0..1
must	7.5.3	0..n
ordered-by	7.7.5	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
type	7.4	1
units	7.3.3	0..1
when	7.19.5	0..1

7.7.3. 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):

- 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.

7.7.4. 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.

7.7.5. 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.

7.7.5.1. 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.

7.7.5.2. 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 7.7.7 for details.

7.7.6. 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 7.7.8 for an example.

7.7.7. 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.

7.7.8. 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
anyxml	7.10	0..n
choice	7.9	0..n
config	7.19.1	0..1
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	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	7.7.4	0..1
min-elements	7.7.3	0..1
must	7.5.3	0..n
ordered-by	7.7.5	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
unique	7.8.3	0..n
uses	7.12	0..n
when	7.19.5	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	7.19.1	0..1
container	7.5	0..n
default	7.9.3	0..1
description	7.19.3	0..1
if-feature	7.18.2	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	7.19.4	0..1
status	7.19.2	0..1
when	7.19.5	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	7.19.3	0..1
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
uses	7.12	0..n
when	7.19.5	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 7.15).

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	7.19.1	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
when	7.19.5	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, in modules that include it, 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
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	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	7.15	0..n
description	7.19.3	0..1
if-feature	7.18.2	0..n
refine	7.12.2	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
when	7.19.5	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:

- A leaf or choice node may get a default value, or a new default value if it already had one.
- Any node may get a specialized "description" string.
- Any node may get a specialized "reference" string.
- Any node may get a different "config" statement.
- A leaf, anyxml, or choice node may get a different "mandatory" statement.
- A container node may get a "presence" statement.
- A leaf, leaf-list, list, container, or anyxml node may get additional "must" expressions.
- A leaf-list or list node may get a different "min-elements" or "max-elements" statement.
- 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 a NETCONF 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	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	0..n
input	7.13.2	0..1
output	7.13.3	0..1
reference	7.19.4	0..1
status	7.19.2	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 RPC operation. It does not take an argument. The substatements to "input" define nodes under the RPC'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 "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
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 RPC'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 "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
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 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 "config" statement is present for any node in the notification tree, the "config" statement is ignored.

7.14.1. The notification's Substatements

substatement	section	cardinality
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
uses	7.12	0..n

7.14.2. 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).

7.14.3. 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>
```

7.15. 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.

7.15.1. The augment's Substatements

substatement	section	cardinality
anyxml	7.10	0..n
case	7.9.2	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
uses	7.12	0..n
when	7.19.5	0..1

7.15.2. 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.

7.15.3. 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>
```

7.16. 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 a base identity. 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.

7.16.1. The identity's Substatements

substatement	section	cardinality
base	7.16.2	0..1
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1

7.16.2. 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 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.

Since submodules cannot include the parent module, any identities in the module that need to be exposed to submodules MUST be defined in a submodule. Submodules can then include this submodule to find the definition of the identity.

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.

7.16.3. 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";
    }
}
```

7.17. 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.

7.17.1. The extension's Substatements

substatement	section	cardinality
argument	7.17.2	0..1
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1

7.17.2. 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.

7.17.2.1. The argument's Substatements

substatement	section	cardinality
yin-element	7.17.2.2	0..1

7.17.2.2. 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".

7.17.3. 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";
  }
}
```

7.18. Conformance-Related Statements

This section defines statements related to conformance, as described in Section 5.6.

7.18.1. 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 7.18.2). 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.

7.18.1.1. The feature's Substatements

substatement	section	cardinality
description	7.19.3	0..1
if-feature	7.18.2	0..n
status	7.19.2	0..1
reference	7.19.4	0..1

7.18.2. 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.

Since submodules cannot include the parent module, any features in the module that need to be exposed to submodules MUST be defined in a submodule. Submodules can then include this submodule to find the definition of the feature.

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.

7.18.2.1. 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";
    ...
}
```

7.18.3. 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.

7.18.3.1. The deviation's Substatements

substatement	section	cardinality
description	7.19.3	0..1
deviate	7.18.3.2	1..n
reference	7.19.4	0..1

7.18.3.2. 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	7.19.1	0..1
default	7.6.4	0..1
mandatory	7.6.5	0..1
max-elements	7.7.4	0..1
min-elements	7.7.3	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

7.18.3.3. 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";
      }
  }
```

7.19. Common Statements

This section defines substatements common to several other statements.

7.19.1. 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".

7.19.2. 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
}
```

7.19.3. 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.

7.19.4. 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";
  ...
}
```

7.19.5. 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.

- 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.

- If the "when" statement is a child of any other data definition statement, the context node is the data definition's node in the data tree.

- The accessible tree is made up of all nodes in the data tree, and all leafs with default values in use (see Section 7.6.1).

The accessible tree depends on the context node:

- If the context node represents configuration, the tree is the data in the NETCONF datastore where the context node exists. The XPath root node has all top-level configuration data nodes in all modules as children.

- If the context node represents state data, the tree is all state data on the device, and the "running" datastore. The XPath root node has all top-level data nodes in all modules as children.

- If the context node represents notification content, the tree is the notification XML instance document. The XPath root node has the element representing the notification being defined as the only child.

- If the context node represents RPC input parameters, the tree is the RPC XML instance document. The XPath root node has the element representing the RPC operation being defined as the only child.

- If the context node represents RPC output parameters, the tree is the RPC reply instance document. The XPath root node has the elements representing the RPC output parameters as children.

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

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.

- If the constraint is defined on configuration data, it MUST be true in a valid configuration data tree.
- If the constraint is defined on state data, it MUST be true in a reply to a <get> operation without a filter.
- If the constraint is defined on notification content, it MUST be true in any notification instance.
- If the constraint is defined on RPC input parameters, it MUST be true in an invocation of the RPC operation.
- 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:

- during parsing of RPC payloads

- during processing of NETCONF operations
- 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.

- 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.
- If all keys of a list entry are not present, the server MUST reply with a "missing-element" error-tag in the rpc-error.
- 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.
- 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.
- 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.
- 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.
- 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:

- Delete requests for non-existent data.
- Create requests for existent data.
- Insert requests with "before" or "after" parameters that do not exist.

During <edit-config> processing:

- If the NETCONF operation creates data nodes under a "choice", any existing nodes from other "case" branches are deleted by the server.
- 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.

- Any "must" constraints MUST evaluate to "true".
- Any referential integrity constraints defined via the "path" statement MUST be satisfied.
- Any "unique" constraints on lists MUST be satisfied.
- 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	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	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 tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [ISO.10646]:

```
;; any Unicode character, excluding the surrogate blocks,
;; FFFE, and FFFF.
string = *char
char = %x9 / %xA / %xD / %x20-D7FF / %xE000-FFFD /
       %x10000-10FFFF
```

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	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	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	7.19.3	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	7.19.4	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 cannot be restricted.

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.

9.6.4.1. The enum's Substatements

substatement	section	cardinality
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	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. The value is unused by YANG and the XML encoding, but is carried as a convenience to implementors.

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.

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>
```

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	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	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) 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 7.18.2), 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:

- The context node is the node in the data tree for which the "path" statement is defined.

The accessible tree depends on the context node:

- If the context node represents configuration data, the tree is the data in the NETCONF datastore where the context node exists. The XPath root node has all top-level configuration data nodes in all modules as children.

- Otherwise, the tree is all state data on the device, and the "running" datastore. The XPath root node has all top-level data nodes in all modules as children.

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 7.16).

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 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 the identityref's base identity. 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. Otherwise, an identity with the matching name MUST be defined in the current module or an included submodule.

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 always present.

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 7.16.3 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.

When a string representing a union data type is validated, the string is validated 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 nodes with their default value in use (see Section 7.6.1) 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:

- The context node is the root node in the accessible tree.

The accessible tree depends on the leaf with the instance-identifier type:

- If this leaf represents configuration data, the tree is the data in the NETCONF datastore where the leaf exists. The XPath root node has all top-level configuration data nodes in all modules as children.

- Otherwise, the tree is all state data on the device, and the "running" datastore. The XPath root node has all top-level data nodes in all modules as children.

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.

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.

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.

10.4.1.1. 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.

- A "bits" type may have new bits added, provided the old bit positions do not change.
- A "range", "length", or "pattern" statement may expand the allowed value space.
- A "default" statement may be added to a leaf that does not have a default value (either directly or indirectly through its type).
- A "units" statement may be added.
- A "reference" statement may be added or updated.
- A "must" statement may be removed or its constraint relaxed.
- A "mandatory" statement may be removed or changed from "true" to "false".
- A "min-elements" statement may be removed, or changed to require fewer elements.
- A "max-elements" statement may be removed, or changed to allow more elements.
- A "description" statement may be added or clarified without changing the semantics of the definition.
- New typedefs, groupings, rpcs, notifications, extensions, features, and identities may be added.
- 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).
- A new "case" statement may be added.
- A node that represented state data may be changed to represent configuration, provided it is not mandatory (Section 3.1).
- An "if-feature" statement may be removed, provided its node is not mandatory (Section 3.1).
- A "status" statement may be added, or changed from "current" to "deprecated" or "obsolete", or from "deprecated" to "obsolete".

- 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.

- 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.

- 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.

- 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 7.17). The following rules hold for arguments:

- If the argument is represented as an attribute, this attribute has no namespace.
- If the argument is represented as an element, it is qualified by the same namespace as its parent keyword element.
- 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 7.17.3, 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] 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 stmtsep
                         namespace-stmt stmtsep
                         prefix-stmt stmtsep

   submodule-header-stmts =
                         ;; these stmts can appear in any order
                         yang-version-stmt stmtsep
                         belongs-to-stmt stmtsep

   meta-stmts          = ;; these stmts can appear in any order
                         [organization-stmt stmtsep]
                         [contact-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   linkage-stmts       = ;; these stmts can appear in any order
                         *(import-stmt stmtsep)
                         *(include-stmt stmtsep)

   revision-stmts      = *(revision-stmt stmtsep)

   body-stmts          = *((extension-stmt /
                            feature-stmt /
                            identity-stmt /
                            typedef-stmt /
                            grouping-stmt /
                            data-def-stmt /
                            augment-stmt /
                            rpc-stmt /
                            notification-stmt /
                            deviation-stmt) stmtsep)

   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
                         optsep 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 stmtsep
                             [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 optsep stmtend

   uri-str             = < a string that matches the rule
                           URI in RFC 3986 >

   prefix-stmt         = prefix-keyword sep prefix-arg-str
                         optsep stmtend

   belongs-to-stmt     = belongs-to-keyword sep identifier-arg-str
                         optsep
                         "{" stmtsep
                             prefix-stmt stmtsep
                         "}"

   organization-stmt   = organization-keyword sep string
                         optsep stmtend

   contact-stmt        = contact-keyword sep string optsep stmtend

   description-stmt    = description-keyword sep string optsep
                         stmtend
```

```
   reference-stmt      = reference-keyword sep string optsep stmtend

   units-stmt          = units-keyword sep string optsep stmtend

   revision-stmt       = revision-keyword sep revision-date optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [description-stmt stmtsep]
                              [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 stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [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
                              [base-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
```

```
                          "}")

   base-stmt           = base-keyword sep identifier-ref-arg-str
                         optsep stmtend

   feature-stmt        = feature-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *(if-feature-stmt stmtsep)
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                          "}")

   if-feature-stmt     = if-feature-keyword sep if-feature-expr-str
                         optsep stmtend

   if-feature-expr-str = < a string that matches the rule
                           if-feature-expr >

   if-feature-expr     = '(' if-fature-expr ')' /
                         if-feature-expr sep boolean-operator sep
                           if-feature-expr /
                         'not' sep if-feature-expr /
                         identifier-ref-arg

   boolean-operator = 'and' / 'or'

   typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             type-stmt stmtsep
                             [units-stmt stmtsep]
                             [default-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                          "}"

   type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                         (";" /
                          "{" stmtsep
                              type-body-stmts
                          "}")

   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 stmtsep

   range-stmt          = range-keyword sep range-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [error-message-stmt stmtsep]
                              [error-app-tag-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                           "}")

   decimal64-specification = ;; these stmts can appear in any order
                             fraction-digits-stmt
                             [range-stmt stmtsep]

   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 stmtsep]
                         *(pattern-stmt stmtsep)

   length-stmt         = length-keyword sep length-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [error-message-stmt stmtsep]
                              [error-app-tag-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                           "}")
```

```
   pattern-stmt        = pattern-keyword sep string optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [modifier-stmt stmtsep]
                              [error-message-stmt stmtsep]
                              [error-app-tag-stmt stmtsep]
                              [description-stmt stmtsep]
                              [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 stmtsep)

   enum-stmt           = enum-keyword sep string optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [value-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                           "}")

   leafref-specification =
                         ;; these stmts can appear in any order
                         path-stmt stmtsep
                         [require-instance-stmt stmtsep]

   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 stmtsep]

   identityref-specification =
                        base-stmt stmtsep

   union-specification = 1*(type-stmt stmtsep)

   binary-specification = [length-stmt stmtsep]

   bits-specification  = 1*(bit-stmt stmtsep)

   bit-stmt            = bit-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [position-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [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 stmtsep]
                              [error-app-tag-stmt stmtsep]
                              [description-stmt stmtsep]
                              [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 stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              *(data-def-stmt stmtsep)
                           "}")

   container-stmt      = container-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [when-stmt stmtsep]
                              *(if-feature-stmt stmtsep)
                              *(must-stmt stmtsep)
                              [presence-stmt stmtsep]
                              [config-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              *(data-def-stmt stmtsep)
                           "}")

   leaf-stmt           = leaf-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             type-stmt stmtsep
                             [units-stmt stmtsep]
                             *(must-stmt stmtsep)
                             [default-stmt stmtsep]
                             [config-stmt stmtsep]
                             [mandatory-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
```

```
                          "}"

   leaf-list-stmt      = leaf-list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             type-stmt stmtsep
                             [units-stmt stmtsep]
                             *(must-stmt stmtsep)
                             [config-stmt stmtsep]
                             [min-elements-stmt stmtsep]
                             [max-elements-stmt stmtsep]
                             [ordered-by-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                          "}"

   list-stmt           = list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; these stmts can appear in any order
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             *(must-stmt stmtsep)
                             [key-stmt stmtsep]
                             *(unique-stmt stmtsep)
                             [config-stmt stmtsep]
                             [min-elements-stmt stmtsep]
                             [max-elements-stmt stmtsep]
                             [ordered-by-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                             *((typedef-stmt /
                                grouping-stmt) stmtsep)
                             1*(data-def-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 stmtsep]
                              *(if-feature-stmt stmtsep)
                              [default-stmt stmtsep]
                              [config-stmt stmtsep]
                              [mandatory-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *((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 stmtsep]
                              *(if-feature-stmt stmtsep)
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *(data-def-stmt stmtsep)
                           "}")

   anyxml-stmt         = anyxml-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [when-stmt stmtsep]
                              *(if-feature-stmt stmtsep)
                              *(must-stmt stmtsep)
                              [config-stmt stmtsep]
```

```
                              [mandatory-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                           "}")

   uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [when-stmt stmtsep]
                              *(if-feature-stmt stmtsep)
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *(refine-stmt stmtsep)
                              *(uses-augment-stmt stmtsep)
                           "}")

   refine-stmt         = refine-keyword sep refine-arg-str optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *(if-feature-stmt stmtsep)
                              *(must-stmt stmtsep)
                              [presence-stmt stmtsep]
                              [default-stmt stmtsep]
                              [config-stmt stmtsep]
                              [mandatory-stmt stmtsep]
                              [min-elements-stmt stmtsep]
                              [max-elements-stmt stmtsep]
                              [description-stmt stmtsep]
                              [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 stmtsep]
                             *(if-feature-stmt stmtsep)
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                             1*((data-def-stmt stmtsep) /
```

```
                              (case-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 stmtsep]
                              *(if-feature-stmt stmtsep)
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              1*((data-def-stmt stmtsep) /
                                 (case-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 stmtsep]
                               [reference-stmt stmtsep]
                            "}")

   rpc-stmt             = rpc-keyword sep identifier-arg-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               *(if-feature-stmt stmtsep)
                               [status-stmt stmtsep]
                               [description-stmt stmtsep]
                               [reference-stmt stmtsep]
                               *((typedef-stmt /
                                  grouping-stmt) stmtsep)
                               [input-stmt stmtsep]
                               [output-stmt stmtsep]
                           "}")

   input-stmt           = input-keyword optsep
```

```
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              1*(data-def-stmt stmtsep)
                          "}"

   output-stmt         = output-keyword optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              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 stmtsep)
                               [status-stmt stmtsep]
                               [description-stmt stmtsep]
                               [reference-stmt stmtsep]
                               *((typedef-stmt /
                                  grouping-stmt) stmtsep)
                               *(data-def-stmt stmtsep)
                           "}")

   deviation-stmt      = deviation-keyword sep
                          deviation-arg-str optsep
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              (deviate-not-supported-stmt /
                                1*(deviate-add-stmt /
                                   deviate-replace-stmt /
                                   deviate-delete-stmt))
                          "}"

   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 optsep
                          (";" /
                           "{" stmtsep
                           "}")

   deviate-add-stmt    = deviate-keyword sep add-keyword-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               [units-stmt stmtsep]
                               *(must-stmt stmtsep)
                               *(unique-stmt stmtsep)
                               [default-stmt stmtsep]
                               [config-stmt stmtsep]
                               [mandatory-stmt stmtsep]
                               [min-elements-stmt stmtsep]
                               [max-elements-stmt stmtsep]
                           "}")

   deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               [units-stmt stmtsep]
                               *(must-stmt stmtsep)
                               *(unique-stmt stmtsep)
                               [default-stmt stmtsep]
                           "}")

   deviate-replace-stmt = deviate-keyword sep replace-keyword-str optsep
                          (";" /
                           "{" stmtsep
                               ;; these stmts can appear in any order
                               [type-stmt stmtsep]
                               [units-stmt stmtsep]
                               [default-stmt stmtsep]
                               [config-stmt stmtsep]
                               [mandatory-stmt stmtsep]
                               [min-elements-stmt stmtsep]
                               [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>

   unknown-statement   = prefix ":" identifier [sep string] optsep
                         (";" / "{" *(yang-stmt stmtsep) "}")

   yang-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 /
                          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 abnfgen's syntax for case-sensitive strings

   ;; statement keywords
   anyxml-keyword      = 'anyxml'
   argument-keyword    = 'argument'
   augment-keyword     = 'augment'
   base-keyword        = 'base'
   belongs-to-keyword  = 'belongs-to'
   bit-keyword         = 'bit'
   case-keyword        = 'case'
   choice-keyword      = 'choice'
   config-keyword      = 'config'
   contact-keyword     = 'contact'
   container-keyword   = 'container'
   default-keyword     = 'default'
   description-keyword = 'description'
   enum-keyword        = 'enum'
   error-app-tag-keyword = 'error-app-tag'
   error-message-keyword = 'error-message'
   extension-keyword   = 'extension'
   deviation-keyword   = 'deviation'
   deviate-keyword     = 'deviate'
   feature-keyword     = 'feature'
   fraction-digits-keyword = 'fraction-digits'
   grouping-keyword    = 'grouping'
   identity-keyword    = 'identity'
   if-feature-keyword  = 'if-feature'
   import-keyword      = 'import'
   include-keyword     = 'include'
   input-keyword       = 'input'
```

```
   key-keyword         = 'key'
   leaf-keyword        = 'leaf'
   leaf-list-keyword   = 'leaf-list'
   length-keyword      = 'length'
   list-keyword        = 'list'
   mandatory-keyword   = 'mandatory'
   max-elements-keyword = 'max-elements'
   min-elements-keyword = 'min-elements'
   modifier-keyword    = 'modifier'
   module-keyword      = 'module'
   must-keyword        = 'must'
   namespace-keyword   = 'namespace'
   notification-keyword= 'notification'
   ordered-by-keyword  = 'ordered-by'
   organization-keyword= 'organization'
   output-keyword      = 'output'
   path-keyword        = 'path'
   pattern-keyword     = 'pattern'
   position-keyword    = 'position'
   prefix-keyword      = 'prefix'
   presence-keyword    = 'presence'
   range-keyword       = 'range'
   reference-keyword   = 'reference'
   refine-keyword      = 'refine'
   require-instance-keyword = 'require-instance'
   revision-keyword    = 'revision'
   revision-date-keyword = 'revision-date'
   rpc-keyword         = 'rpc'
   status-keyword      = 'status'
   submodule-keyword   = 'submodule'
   type-keyword        = 'type'
   typedef-keyword     = 'typedef'
   unique-keyword      = 'unique'
   units-keyword       = 'units'
   uses-keyword        = 'uses'
   value-keyword       = 'value'
   when-keyword        = 'when'
   yang-version-keyword= 'yang-version'
   yin-element-keyword = 'yin-element'

   ;; other keywords

   add-keyword         = 'add'
   current-keyword     = 'current'
   delete-keyword      = 'delete'
   deprecated-keyword  = 'deprecated'
   false-keyword       = 'false'
   invert-match-keyword = 'invert-match'
```

```
   max-keyword         = 'max'
   min-keyword         = 'min'
   not-supported-keyword = 'not-supported'
   obsolete-keyword    = 'obsolete'
   replace-keyword     = 'replace'
   system-keyword      = 'system'
   true-keyword        = 'true'
   unbounded-keyword   = 'unbounded'
   user-keyword        = 'user'

   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  = [prefix ":"] identifier

   string              = < an unquoted string as returned by
                           the scanner >

   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             = ";" / "{" *unknown-statement "}"
```

```
   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:

- the name of the module or submodule
- for modules, the assigned XML namespace
- for modules, the prefix of the module
- for submodules, the name of the module it belongs to
- 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 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:

- the security of the transmission infrastructure used to send sensitive information.
- the security of applications that store or release such sensitive information.
- 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 -01

- Included solution Y01-01.
- Included solution Y03-01.
- Included solution Y06-02.
- Included solution Y07-01.
- Included solution Y14-01.
- Included solution Y20-01.
- Included solution Y23-01.
- Included solution Y29-01.
- Included solution Y30-01.
- Included solution Y31-01.
- Included solution Y35-01.

19.2. Version -00

- Applied all reported errata for RFC 6020.
- Updated YANG version to 1.1, and made the "yang-version" statement mandatory.

20. References

20.1. Normative References

[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.

[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

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 April 26, 2015.

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:

- capabilities
- client
- operation
- server

2.3. YANG Terms

The following terms are defined in [RFC6020] and are not redefined here:

- data node
- module
- namespace
- submodule
- version
- YANG
- 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:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- 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:

- Narrative sections
- Definitions section
- Security Considerations section
- IANA Considerations section
- 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:

- Writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be explained.

- 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.

- 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:

- the module to compile correctly instead of generating disruptive fatal errors.
- early implementors to use the modules without picking a random value for the XML namespace.
- 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:

- anyxml
- augment
- choice
- container
- extension
- feature
- grouping
- identity
- leaf
- leaf-list

- list
- notification
- rpc
- 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 '' and '', 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:

- 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.

- 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.

- 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/

- 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.

- 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].

- 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).

- 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/

- Other Issues -- check for any issues mentioned in http://www.ietf.org/id-info/checklist.html that are not covered elsewhere.

- 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: April 29, 2015

L. Lhotka
CZ.NIC

October 26, 2014

A YANG Data Model for Routing Management
draft-ietf-netmod-routing-cfg-16

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), routing protocols and route filters.

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 April 29, 2015.

Copyright Notice

Table of Contents

1. Introduction

This document contains a specification of the following YANG modules:

- Module "ietf-routing" provides generic components of a routing data model.

- Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.

- 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]:

- client
- message
- protocol operation
- server

The following terms are defined in [RFC6020]:

- augment
- configuration data
- data model
- data node
- feature
- mandatory node
- module
- state data
- RPC operation

2.1. Glossary of New Terms

active route: a route that is actually used for sending packets. If there are multiple candidate routes with a matching destination prefix, then it is up to the routing algorithm to select the active route.

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:

- Brackets "[" and "]" enclose list keys.
- Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- Abbreviations before data node names: "rw" means configuration (read-write), and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node and "*" denotes a "list" or "leaf-list".
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- 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 methods 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:

- 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).

- Simple routing set-ups, such as static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.

- 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.

- 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]
      |              ...
      +--rw route-filters
         +--rw route-filter* [name]
            +--rw name
            +--rw description?
            +--rw type
```

Figure 1: Configuration data hierarchy.

```
   +--ro routing-state
      +--ro routing-instance* [name]
      |  +--ro name
      |  +--ro id
      |  +--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 next-hop-lists
      |  +--ro next-hop-list* [id]
      |     +--ro id
      |     +--ro address-family
      |     +--ro next-hop*
      |        +--ro (next-hop-options)
      |        |     ...
      |        +--ro priority?
      |        +--ro weight?
      +--ro ribs
      |  +--ro rib* [name]
      |     +--ro name
      |     +--ro id
      |     +--ro address-family
      |     +--ro routes
      |     |  +--ro route*
      |     |        ...
      |     +--ro recipient-ribs
      |        +--ro recipient-rib* [rib-name]
      |              ...
      +--ro route-filters
         +--ro route-filter* [name]
            +--ro name
            +--ro type
```

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, routing protocols and route filters. 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.

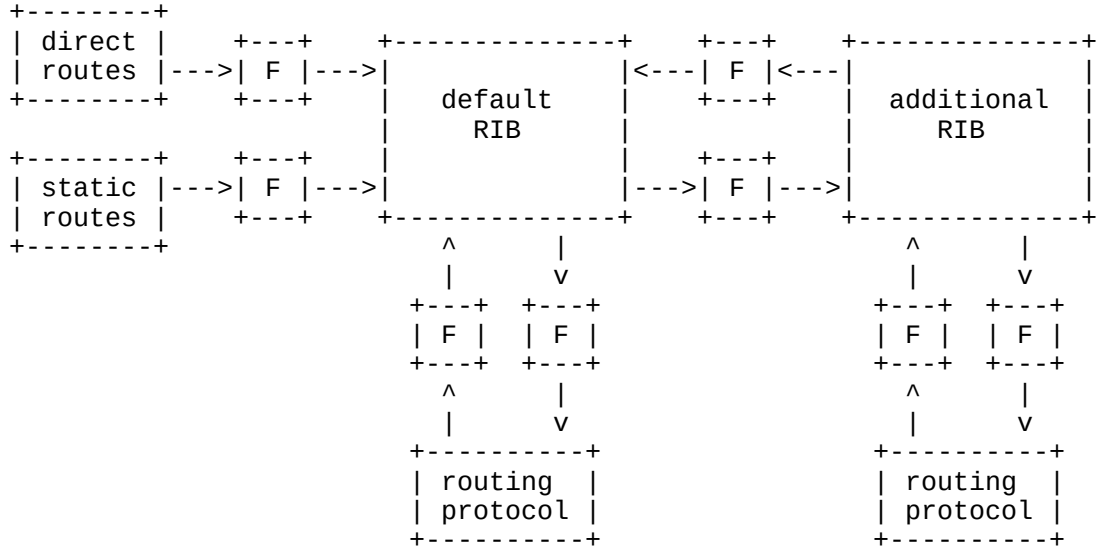


Figure 3: Example set-up of a routing system

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:

- Along with the default RIB, which is always present, an additional RIB is configured.
- Each routing protocol instance, including the "static" and "direct" pseudo-protocols, is connected to exactly one RIB with which it can exchange routes (in both directions, except for the "static" and "direct" pseudo-protocols).
- RIBs may also be connected to each other and exchange routes in either direction (or both).

- o Route exchanges along all connections may be controlled by means of route filters, denoted by "F" in Figure 3.

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.

4.2. Features of Advanced Routers

The core routing data model attempts to address devices with elementary routing functions as well as advanced routers. For simple devices, some parts and options of the data model are not needed and would represent unnecessary complications for the implementation. Therefore, the core routing data model makes the configuration of

some advanced functions optional to implement by means of two YANG features:

- "multiple-ribs" - indicates that the device supports configuration of user-defined RIBs, routing protocols connected to non-default RIBs, and RIBs configured as receivers of routes from other RIBs.

- "multipath-routes" - indicates that the device supports configuration of routes with multiple next-hops.

See the "ietf-routing" module for details.

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:

- send-advertisements,
- max-rtr-adv-interval,
- min-rtr-adv-interval,
- managed-flag,
- other-config-flag,
- link-mtu,
- reachable-time,
- retrans-timer,
- cur-hop-limit,
- default-lifetime,
- 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:

- "destination-prefix": IP prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.

- "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.

- "next-hop": determines the action to be performed with a packet. See below for details.

The choice of next-hops comprises the following cases:

- simple next-hop - IP address of the next-hop router, outgoing interface, or both.

- 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.

- next-hop list reference - each next-hop list is a set of next-hops that may also contain a reference to another next-hop list.

- RIB reference - a new look-up is to be performed in the specified RIB.

It is expected that future modules defining routing protocols will add other route attributes such as metrics or preferences.

Routes are primarily state data that appear as entries of RIBs (Section 5.3) but they may be also 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:

- "source-protocol": type of the routing protocol from which the route was originally obtained.

- "preferred": 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.

- "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. In the data model, address family is represented with 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. Optionally, a route filter may be configured for any or all recipient RIBs. Such a route filter then selects and/or manipulates the routes that are passed between the source and recipient 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 are permitted.

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:

- Routes learned from the network are installed in all connected RIBs with a matching address family.

- 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.

In addition, two independent route filters (see Section 5.5) may be configured for each connected RIB to apply user-defined policies controlling the exchange of routes in both directions between the routing protocol instance and the connected RIB:

- import filter controls which routes are passed from the routing protocol instance to the connected RIB,

- export filter controls which routes the routing protocol instance receives from the connected RIB.

Note that the terms import and export are used from the viewpoint of a RIB.

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 (subject to route filters, if any), 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. Direct routes MAY be filtered before they appear in the default RIB.

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.

Static routes are configured within the "static-routes" container, see Figure 4.

```
+--rw static-routes
   +--rw v4ur:ipv4
   |  +--rw v4ur:route* [destination-prefix]
   |     +--rw v4ur:destination-prefix
   |     +--rw v4ur:description?
   |     +--rw v4ur:next-hop
   |        +--rw (simple-or-list)?
   |           +--:(multipath-entry)
   |           |  +--rw v4ur:multipath-entry* [name]
   |           |     +--rw v4ur:name
   |           |     +--rw (next-hop-options)
   |           |     |  +--:(simple-next-hop)
   |           |     |  |  +--rw v4ur:outgoing-interface?
   |           |     |  +--:(special-next-hop)
   |           |     |  |  +--rw v4ur:special-next-hop?
   |           |     |  +--:(next-hop-address)
   |           |     |     +--rw v4ur:next-hop-address?
   |           |     +--rw v4ur:priority?
   |           |     +--rw v4ur:weight?
   |           +--:(simple-next-hop)
   |              +--rw (next-hop-options)
   |                 +--:(simple-next-hop)
   |                 |  +--rw v4ur:outgoing-interface?
   |                 +--:(special-next-hop)
   |                 |  +--rw v4ur:special-next-hop?
   |                 +--:(next-hop-address)
   |                    +--rw v4ur:next-hop-address?
   |
   +--rw v6ur:ipv6
      +--rw v6ur:route* [destination-prefix]
         +--rw v6ur:destination-prefix
         +--rw v6ur:description?
         +--rw v6ur:next-hop
            +--rw (simple-or-list)?
               +--:(multipath-entry)
               |  +--rw v6ur:multipath-entry* [name]
               |     +--rw v6ur:name
               |     +--rw (next-hop-options)
               |     |  +--:(simple-next-hop)
               |     |  |  +--rw v6ur:outgoing-interface?
               |     |  +--:(special-next-hop)
```

```
|     |  |  +--rw v6ur:special-next-hop?
|     |  +--:(next-hop-address)
|     |     +--rw v6ur:next-hop-address?
|     +--rw v6ur:priority?
|     +--rw v6ur:weight?
+--:(simple-next-hop)
   +--rw (next-hop-options)
      +--:(simple-next-hop)
      |  +--rw v6ur:outgoing-interface?
      +--:(special-next-hop)
      |  +--rw v6ur:special-next-hop?
      +--:(next-hop-address)
         +--rw v6ur:next-hop-address?
```

Figure 4: Structure of "static-routes" subtree.

A next-hop in static routes may be configured as a simple next-hop (IP address, outgoing interface or both), special next-hop or a list of multi-path next-hop entries that is used either for backup routes of for equal-cost multi-path (ECMP) routing. The last option is available only on devices that advertise the feature "rt:multipath-routes". Moreover, unlike next-hop lists in state data, a list of next-hop entries in a static route cannot be recursive, i.e., each entry of that list can only be a simple or special next-hop.

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 and RPC input or output.

- 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".

- 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. Route Filter

The core routing data model provides a skeleton for defining route filters that can be used to restrict the set of routes being exchanged between a routing protocol instance and a connected RIB, or between a source and a recipient RIB. Route filters may also manipulate routes, i.e., add, delete, or modify their attributes.

Route filters are global, which means that a configured route filter may be used by any or all routing instances. However, a data model for a routing instance type MAY specify rules and restrictions for sharing route filters among routing instances of that type.

The core routing data model defines only two extreme route filtering policies which are represented by the following pre-defined route filter types:

- "deny-all-route-filter": all routes are blocked,

- "allow-all-route-filter": all routes are permitted.

The latter type is equivalent to no route filter.

It is expected that more comprehensive route filtering frameworks will be developed separately.

Each route filter entry is identified by a unique name. Its type MUST be specified by the "type" identity reference.

5.6. RPC Operations

The "ietf-routing" module defines two RPC operations:

- 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.

- 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 "routing@2014-10-26.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 2014-10-26 {
      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 advertize 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 multipath-routes {
      description
```

```
        "This feature indicates that the server supports multipath
         routes that have a list of next-hops.";
    }

    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 advertize 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 which provides routes to directly
          connected networks.";
     }

     identity static {
       base routing-protocol;
       description
         "Static routing pseudo-protocol.";
     }

     identity route-filter {
       description
         "Base identity from which all route filters are derived.";
     }

     identity deny-all-route-filter {
       base route-filter;
       description
         "Route filter that blocks all routes.";
     }

     identity allow-all-route-filter {
       base route-filter;
       description
         "Route filter that permits all routes.";
     }

     /* 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 next-hop-list-ref {
      type leafref {
        path "/rt:routing-state/rt:next-hop-lists/rt:next-hop-list/"
           + "rt:id";
      }
      description
        "This type is used for leafs that reference a next-hop list (in
         state data).";
    }

    typedef route-filter-ref {
      type leafref {
        path "/rt:routing/rt:route-filters/rt:route-filter/rt:name";
      }
      description
        "This type is used for leafs that reference a route filter
         configuration.";
    }

    typedef route-filter-state-ref {
      type leafref {
        path "/rt:routing-state/rt:route-filters/rt:route-filter/"
```

```
          + "rt:name";
      }
      description
        "This type is used for leafs that reference state data of a
         route filter.";
    }

    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 state-entry-id {
      description
        "This grouping provides a unique identifier for entries in
         several operational state lists.";
      leaf id {
        type uint64;
        description
          "Unique numerical identifier of a list entry in operational
           state. It may be used by protocols or tools that inspect
           and/or manipulate operational state data and prefer
           fixed-size integers as list entry handles.

           These identifiers are always ephemeral, i.e., they may
           change after a reboot.";
      }
    }

    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 next-hop-classifiers {
      description
        "This grouping provides two next-hop classifiers.";
      leaf priority {
        type enumeration {
          enum primary {
            value "1";
            description
              "Primary next-hop.";
          }
          enum backup {
            value "2";
            description
              "Backup next-hop.";
          }
        }
        description
          "Simple priority for distinguishing between primary and
           backup next-hops.

           Backup next-hops are used if and only if no primary
           next-hops are reachable.";
      }
      leaf weight {
        type uint8;
        must ". = 0 or not(../../next-hop/weight = 0)" {
          error-message "Illegal combination of zero and non-zero "
                      + "next-hop weights.";
          description
            "Next-hop weights must be either all zero (equal
             load-balancing) or all non-zero.";
        }
        description
          "This parameter specifies the weight of the next-hop for load
           balancing. The number specifies the relative fraction of the
           traffic that will use the corresponding next-hop.
```

```
            A value of 0 represents equal load-balancing.

            If both primary and backup next-hops are present, then the
            weights for each priority level are used separately.";
        }
      }

      grouping special-next-hop {
        description
          "This grouping provides a leaf with 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.";
          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.";
        leaf next-hop-list {
          type next-hop-list-ref;
          description
            "Reference to a next-hop list.";
        }
        leaf use-rib {
          type rib-state-ref;
          description
            "Reference to a RIB in which a new look-up is to be
             performed.";
        }
        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
        "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.";
      }
    }

    /* Operational state data */

    container routing-state {
      config "false";
      description
        "Operational state of the routing subsystem.";
      list routing-instance {
        key "name";
```

```
        unique "id";
        min-elements "1";
        description
          "Each list entry is a container for operational 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.";
        }
        uses state-entry-id {
          refine "id" {
            mandatory "true";
          }
        }
        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
            "Operational state 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.";
            }
            leaf import-filter {
              type route-filter-state-ref;
              description
                "Reference to a route filter that is used for
                 filtering routes passed from this routing protocol
                 instance to the RIB specified by the 'rib-name'
                 sibling node.

                 If this leaf is not present, the behavior is
                 protocol-specific, but typically it means that all
                 routes are accepted.";
            }
            leaf export-filter {
              type route-filter-state-ref;
              description
                "Reference to a route filter that is used for
                 filtering routes passed from the RIB specified by
```

```
                    the 'rib-name' sibling node to this routing
                    protocol instance.

                    If this leaf is not present, the behavior is
                    protocol-specific - typically it means that all
                    routes are accepted.

                    The 'direct' and 'static' pseudo-protocols accept
                    no routes from any RIB.";
                }
              }
            }
          }
        }
      }
      container next-hop-lists {
        description
          "Container for next-hop lists.";
        list next-hop-list {
          key "id";
          description
            "Next-hop list.";
          uses state-entry-id;
          uses address-family;
          list next-hop {
            description
              "Entry in a next-hop list.";
            uses next-hop-state-content;
            uses next-hop-classifiers;
          }
        }
      }
      container ribs {
        description
          "Container for RIBs.";
        list rib {
          key "name";
          unique "id";
          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 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 state-entry-id {
            refine "id" {
              mandatory "true";
            }
          }
          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.";
              }
              leaf filter {
                type route-filter-state-ref;
```

```
                description
                  "A route filter which is applied to the routes passed
                   to the recipient RIB.";
              }
            }
          }
        }
      }
      container route-filters {
        description
          "Container for route filters.";
        list route-filter {
          key "name";
          description
            "Route filters are used for filtering and/or manipulating
             routes that are passed between a routing protocol and a
             RIB and vice versa, or between two RIBs.

             It is expected that other modules augment this list with
             contents specific for a particular route filter type.";
          leaf name {
            type string;
            description
              "The name of the route filter.";
          }
          leaf type {
            type identityref {
              base route-filter;
            }
            mandatory "true";
            description
              "Type of the route-filter - an identity derived from the
               'route-filter' base identity.";
          }
        }
      }
    }

    /* 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 operational 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
                 operational 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.";
                }
                leaf import-filter {
                  type route-filter-ref;
                  description
                    "Configuration of import filter.";
                }
                leaf export-filter {
                  type route-filter-ref;
                  description
                    "Configuration of export filter.";
                }
              }
            }
            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.";
            }
            leaf filter {
              type route-filter-ref;
              description
                "A route filter which is applied to the routes passed
                 to the recipient RIB.";
            }
          }
        }
      }
    }
    container route-filters {
      description
        "Configuration of route filters.";
      list route-filter {
        key "name";
        description
          "Each entry configures a named route filter.";
        leaf name {
          type string;
          description
            "The name of the route filter.";
        }
        leaf description {
          type string;
          description
            "Textual description of the route filter.";
        }
        leaf type {
          type identityref {
            base route-filter;
          }
          mandatory "true";
          description
            "Type of the route filter..";
        }
      }
    }
  }

  /* RPC methods */

  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 route for the specified destination.

             If the routing instance has no active 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 "ipv4-unicast-routing@2014-10-26.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 operational 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 2014-10-26 {
  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.";
    }

    /* Operational 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.";
      }
    }

    augment "/rt:routing-state/rt:next-hop-lists/rt:next-hop-list/"
          + "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 next-hop list with IPv4 next-hop address.
         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.";
          grouping next-hop-content {
            description
              "Next-hop content for IPv4 unicast static routes.";
            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.";
                }
              }
            }
          }
```

```
            choice simple-or-list {
              description
                "Options for next-hops.";
              list multipath-entry {
                if-feature rt:multipath-routes;
                key "name";
                description
                  "List of alternative next-hops.";
                leaf name {
                  type string;
                  description
                    "A unique identifier of the next-hop entry.";
                }
                uses next-hop-content;
                uses rt:next-hop-classifiers;
              }
              case simple-next-hop {
                uses next-hop-content;
              }
            }
          }
        }
      }

      /* RPC methods */

      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 "ipv6-unicast-routing@2014-10-26.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 operational 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 2014-10-26 {
    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.";
    }

    /* Operational 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 {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:next-hop-lists/rt:next-hop-list/"
          + "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 next-hop list with IPv6 next-hop address.
         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.";
            grouping next-hop-content {
              description
                "Next-hop content for IPv6 unicast static routes.";
              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.";
                  }
                }
              }
            }
            choice simple-or-list {
              description
                "Options for next-hops.";
              list multipath-entry {
                if-feature rt:multipath-routes;
                key "name";
                description
                  "List of alternative next-hops.";
                leaf name {
                  type string;
                  description
                    "A unique identifier of the next-hop entry.";
                }
                uses next-hop-content;
                uses rt:next-hop-classifiers;
              }
              case simple-next-hop {
                uses next-hop-content;
              }
            }
          }
        }
      }

      /* RPC methods */

      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='v4ur: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/route-filters/route-filter: This list specifies the configured route filters which represent administrative policies for redistributing and modifying routing information.

/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 author wishes to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Acee Lindem, 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 import-filter?   route-filter-ref
|        |     +--rw export-filter?   route-filter-ref
|        +--rw static-routes
|           +--rw v4ur:ipv4
|           |  +--rw v4ur:route* [destination-prefix]
|           |     +--rw v4ur:destination-prefix inet:ipv4-prefix
|           |     +--rw v4ur:description?          string
|           |     +--rw v4ur:next-hop
|           |        +--rw (simple-or-list)?
|           |           +--:(multipath-entry)
|           |           |  +--rw v4ur:multipath-entry* [name]
|           |           |     +--rw v4ur:name             string
|           |           |     +--rw (next-hop-options)
|           |           |     |  +--:(simple-next-hop)
|           |           |     |  |  +--rw v4ur:outgoing-interface?
```

```
      |           |           |     |  +--:(special-next-hop)
      |           |           |     |  |  +--rw v4ur:special-next-hop?
      |           |           |     |  +--:(next-hop-address)
      |           |           |     |     +--rw v4ur:next-hop-address?
      |           |           |     +--rw v4ur:priority?
      |           |           |     +--rw v4ur:weight?           uint8
      |           |           +--:(simple-next-hop)
      |           |              +--rw (next-hop-options)
      |           |                 +--:(simple-next-hop)
      |           |                 |  +--rw v4ur:outgoing-interface?
      |           |                 +--:(special-next-hop)
      |           |                 |  +--rw v4ur:special-next-hop?
      |           |                 +--:(next-hop-address)
      |           |                    +--rw v4ur:next-hop-address?
      |           +--rw v6ur:ipv6
      |              +--rw v6ur:route* [destination-prefix]
      |                 +--rw v6ur:destination-prefix inet:ipv6-prefix
      |                 +--rw v6ur:description?         string
      |                 +--rw v6ur:next-hop
      |                    +--rw (simple-or-list)?
      |                       +--:(multipath-entry)
      |                       |  +--rw v6ur:multipath-entry* [name]
      |                       |     +--rw v6ur:name             string
      |                       |     +--rw (next-hop-options)
      |                       |     |  +--:(simple-next-hop)
      |                       |     |  |  +--rw v6ur:outgoing-interface?
      |                       |     |  +--:(special-next-hop)
      |                       |     |  |  +--rw v6ur:special-next-hop?
      |                       |     |  +--:(next-hop-address)
      |                       |     |     +--rw v6ur:next-hop-address?
      |                       |     +--rw v6ur:priority?
      |                       |     +--rw v6ur:weight?           uint8
      |                       +--:(simple-next-hop)
      |                          +--rw (next-hop-options)
      |                             +--:(simple-next-hop)
      |                             |  +--rw v6ur:outgoing-interface?
      |                             +--:(special-next-hop)
      |                             |  +--rw v6ur:special-next-hop?
      |                             +--:(next-hop-address)
      |                                +--rw v6ur: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
```

```
      |           +--rw filter?     route-filter-ref
      +--rw route-filters
         +--rw route-filter* [name]
            +--rw name           string
            +--rw description?   string
            +--rw type           identityref
```

A.2. State Data

```
   +--ro routing-state
      +--ro routing-instance* [name]
      |  +--ro name                 string
      |  +--ro id                   uint64
      |  +--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 import-filter?   route-filter-state-ref
```

```
|              +--ro export-filter?   route-filter-state-ref
+--ro next-hop-lists
|  +--ro next-hop-list* [id]
|     +--ro id                uint64
|     +--ro address-family    identityref
|     +--ro next-hop*
|        +--ro (next-hop-options)
|        |  +--:(next-hop-list)
|        |  |  +--ro next-hop-list?           next-hop-list-ref
|        |  +--:(use-rib)
|        |  |  +--ro use-rib?                 rib-state-ref
|        |  +--:(simple-next-hop)
|        |  |  +--ro outgoing-interface?
|        |  |  +--ro v4ur:next-hop-address?   inet:ipv4-address
|        |  |  +--ro v6ur:next-hop-address?   inet:ipv6-address
|        |  +--:(special-next-hop)
|        |     +--ro special-next-hop?        enumeration
|        +--ro priority?                enumeration
|        +--ro weight?                  uint8
+--ro ribs
|  +--ro rib* [name]
|     +--ro name              string
|     +--ro id                uint64
|     +--ro address-family    identityref
|     +--ro routes
|     |  +--ro route*
|     |     +--ro route-preference?          route-preference
|     |     +--ro next-hop
|     |     |  +--ro (next-hop-options)
|     |     |     +--:(next-hop-list)
|     |     |     |  +--ro next-hop-list?      next-hop-list-ref
|     |     |     +--:(use-rib)
|     |     |     |  +--ro use-rib?                rib-state-ref
|     |     |     +--:(simple-next-hop)
|     |     |     |  +--ro outgoing-interface?
|     |     |     |  +--ro v4ur:next-hop-address?
|     |     |     |  +--ro v6ur:next-hop?
|     |     |     +--:(special-next-hop)
|     |     |        +--ro special-next-hop?        enumeration
|     |     +--ro source-protocol            identityref
|     |     +--ro active?                    empty
|     |     +--ro last-updated?              yang:date-and-time
|     |     +--ro v4ur:destination-prefix?   inet:ipv4-prefix
|     |     +--ro v6ur:destination-prefix?   inet:ipv6-prefix
|     +--ro recipient-ribs
|        +--ro recipient-rib* [rib-name]
|           +--ro rib-name    rib-state-ref
|           +--ro filter?     route-filter-state-ref
```

```
      +--ro route-filters
         +--ro route-filter* [name]
            +--ro name    string
            +--ro type    identityref
```

Appendix B. Minimum Implementation

Some parts and options of the core routing model, such as route filters or multiple 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, neither of the features defined in the "ietf-routing" module ("multiple-ribs" and "multipath-routes") will be supported. This means that:

- A single system-controlled RIB (routing table) is available for each supported address family - IPv4, IPv6 or both. These RIBs are 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.

- Each route has no more than one "next-hop", "outgoing-interface" or "special-next-hop".

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 configuring appropriate route filters ("allow-all-route-filter" or "deny-all-route-filter").

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" routing protocol instances, preventing any route filters to be configured etc.

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 5: 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.

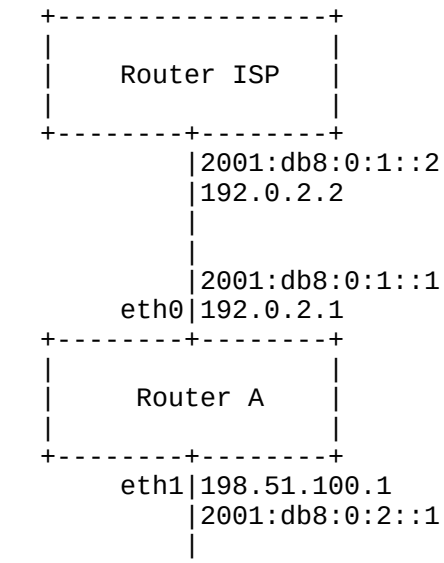


Figure 5: Example network configuration

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:id>2718281828</rt:id>
```

```
      <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:id>897932384</rt:id>
       <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:id>751058209</rt:id>
       <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>2001:db8:0:1::2</v6ur:next-hop>
         </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 -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.
- Remove the ’id’ key from static routes and make ’destination-prefix’ the only key.

- Added ’route-preference’ as a new attribute of routes in RIB.
- Added ’active’ as a new attribute of routes in RIBs.
- Renamed RPC operation ’active-route’ to ’fib-route.
- Added ’route-preference’ as a new parameter of routing protocol instances, both in configuration and state data.
- Renamed identity ’rt:standard-routing-instance’ to ’rt:default-routing-instance’.
- Added next-hop lists to state data.
- Added two cases for specifying next-hops indirectly - via a new RIB or a recursive list of next-hops.
- Reorganized next-hop in static routes.
- Removed all ’if-feature’ statements from state data.

E.2. Changes Between Versions -14 and -15

- Removed all defaults from state data.
- Removed default from ’cur-hop-limit’ in config.

E.3. Changes Between Versions -13 and -14

- Removed dependency of ’connected-ribs’ on the ’multiple-ribs’ feature.
- Removed default value of ’cur-hop-limit’ in state data.
- Moved parts of descriptions and all references on IPv6 RA parameters from state data to configuration.
- Added reference to RFC 6536 in the Security section.

E.4. Changes Between Versions -12 and -13

- Wrote appendix about minimum implementation.
- Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.
- Extra container for the next-hop list.

- Names rather than numeric ids are used for referring to list entries in state data.
- Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- Descriptions of "name" keys in state data lists are required to be persistent.
-
- Removed "if-feature multiple-ribs;" from connected-ribs.
- "rib-name" instead of "name" is used as the name of leafref nodes.
- "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

E.5. Changes Between Versions -11 and -12

- Removed feature "advanced-router" and introduced two features instead: "multiple-ribs" and "multipath-routes".
- Unified the keys of config and state versions of "routing-instance" and "rib" lists.
- Numerical identifiers of state list entries are not keys anymore, but they are constrained using the "unique" statement.
- Updated acknowledgements.

E.6. Changes Between Versions -10 and -11

- Migrated address families from IANA enumerations to identities.
- Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- Described the relationship between system-controlled and user-controlled list entries.
- Feature "user-defined-routing-tables" changed into "advanced-router".

- Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).
- Added nexthop-list with entries having priorities (backup) and weights (load balancing).
- Updated bibliography references.

E.7. Changes Between Versions -09 and -10

- Added subtree for state data ("/routing-state").
- Terms "system-controlled entry" and "user-controlled entry" defined and used.
- New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.
- Added grouping "router-id".
- In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".
- Renamed "main-routing-table" to "default-routing-table".

E.8. Changes Between Versions -08 and -09

- Fixed "must" expresion for "connected-routing-table".
- Simplified "must" expression for "main-routing-table".
- Moved per-interface configuration of a new routing protocol under ’routing-protocol’. This also affects the ’example-rip’ module.

E.9. Changes Between Versions -07 and -08

- Changed reference from RFC6021 to RFC6021bis.

E.10. Changes Between Versions -06 and -07

- 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.
- 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.

- The default behavior for IPv6 RA prefix advertisements was clarified.
- Changed type of "rt:router-id" to "ip:dotted-quad".
- Type of "rt:router-id" changed to "yang:dotted-quad".
- Fixed missing prefixes in XPath expressions.

E.11. Changes Between Versions -05 and -06

- Document title changed: "Configuration" was replaced by "Management".
- New typedefs "routing-table-ref" and "route-filter-ref".
- Double slashes "//" were removed from XPath expressions and replaced with the single "/".
- Removed uniqueness requirement for "router-id".
- Complete data tree is now in Appendix A.
- Changed type of "source-protocol" from "leafref" to "string".
- Clarified the relationship between routing protocol instances and connected routing tables.
- Added a must constraint saying that a routing table connected to the direct pseudo-protocol must not be a main routing table.

E.12. Changes Between Versions -04 and -05

- Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- "must" statement for "static-routes" changed to "when".
- Added "main-routing-tables" containing references to main routing tables for each address family.
- Removed the defaults for "address-family" and "safi" and made them mandatory.
- Removed the default for route-filter/type and made this leaf mandatory.

- If there is no active route for a given destination, the "active-route" RPC returns no output.
- Added "enabled" switch under "routing-protocol".
- Added "router-type" identity and "type" leaf under "router".
- Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- The "direct" pseudo-protocol is always connected to main routing tables.
- Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".
- Added "must" constraint saying that a routing table must not be its own recipient.

E.13. Changes Between Versions -03 and -04

- Changed "error-tag" for both RPC methods from "missing element" to "data-missing".
- Removed the decrementing behavior for advertised IPv6 prefix parameters "valid-lifetime" and "preferred-lifetime".
- Changed the key of the static route lists from "seqno" to "id" because the routes needn’t be sorted.
- Added ’must’ constraint saying that "preferred-lifetime" must not be greater than "valid-lifetime".

E.14. Changes Between Versions -02 and -03

- Module "iana-afn-safi" moved to I-D "iana-if-type".
- Removed forwarding table.
- RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- New RPC "route-count".
- For both RPCs, specification of negative responses was added.
- Relaxed separation of router instances.

- Assignment of interfaces to router instances needn’t be disjoint.
- Route filters are now global.
- Added "allow-all-route-filter" for symmetry.
- Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- Added "router-id" leaf.
- Specified the names for IPv4/IPv6 unicast main routing tables.
- Route parameter "last-modified" changed to "age".
- Added container "recipient-routing-tables".

E.15. Changes Between Versions -01 and -02

- Added module "ietf-ipv6-unicast-routing".
- The example in Appendix D now uses IP addresses from blocks reserved for documentation.
- Direct routes appear by default in the forwarding table.
- Network layer interfaces must be assigned to a router instance. Additional interface configuration may be present.
- The "when" statement is only used with "augment", "must" is used elsewhere.
- Additional "must" statements were added.
- The "route-content" grouping for IPv4 and IPv6 unicast now includes the material from the "ietf-routing" version via "uses rt:route-content".
- Explanation of symbols in the tree representation of data model hierarchy.

E.16. Changes Between Versions -00 and -01

- AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.

- Names of some data nodes were changed, in particular "routing-process" is now "router".
- The restriction of a single AFN/SAFI per router was lifted.
- RPC operation "delete-route" was removed.
- Illegal XPath references from "get-route" to the datastore were fixed.
- Section "Security Considerations" was written.

Author’s Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 15, 2015

L. Lhotka
CZ.NIC

September 11, 2014

Defining and Using Metadata with YANG
draft-lhotka-netmod-yang-metadata-00

Abstract

This document defines a YANG extension statement that allows for defining metadata annotions in YANG modules. The document also specifies the encoding of annotations and rules for annotating instances of YANG data nodes.

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 March 15, 2015.

Copyright Notice

Table of Contents

1. Introduction

There is a need to be able to annotate instances of YANG [3] data nodes with various metadata. Typical use cases are:

- Deactivating a subtree in a configuration datastore while keeping the data in place.
- Qualifying the data model information with instance-specific data. For example, an annotation may be attached to an instance of a leaf with the "union" type to indicate the member type to which the instance belongs.
- RPC operations may use metadata annotations for different 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 and using metadata annotations that satisfies the above requirements. 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 one or more annotations.

The benefits of defining metadata annotations in a YANG module are as follows:

- 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 support for certain annotations using standard module advertisement methods, such as the <hello> message in NETCONF.

- Values of annotations need not be 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,

- datastore,

- message,
- operation,
- server.

The following terms are defined in [3]:

- anyxml,
- built-in type,
- derived type,
- container,
- data model,
- data node,
- derived type,
- extension,
- leaf-list,
- list,
- module,
- RPC operation,
- submodule,
- type.

The following terms are defined in [8]:

- attribute,
- document,
- element,
- namespace,
- prefix.

The following terms are defined in [6]:

- array,
- member,
- object,
- primitive type.

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
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.

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,
       and its value is 'true', then the server MUST behave
       as if the data subtree rooted at this node was not
       present.";
  }
}
```

Metadata annotations defined with the "md:annotation" statement may be attached to any valid instance of a data node, i.e., container, leaf, list, leaf-list or anyxml, throughout the data model. Metadata annotations are always optional.

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:

- 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.

- 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 as 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 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 (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, Anyxml and List Instances

For an instance that is translated to a JSON object (i.e., a container, anyxml or list entry), the metadata object is added as a new member of that object with the name "@".

Examples:

- "cask" is a container or anyxml node:

```
"cask": {
  "@": {
    "example-inactive:inactive": true
  },
  ...
}
```

- "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 Instances

For a leaf instance, the metadata object is added as a sibling name/value pair whose the name is the symbol "@" concatenated with the identifier of the leaf.

For example, if "flag" is a leaf node:

```
"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 identifier of the leaf-list, 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:

```
"folio": [6, 3, 7, 8],
"@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 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. The "rng:attribute" pattern 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:attribute name="ein:inactive">
    <rng:choice>
      <rng:value>true</rng:value>
      <rng:value>false</rng:value>
    </rng:choice>
  </rng:attribute>
</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 Ed.: 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 "yang-metadata@2014-09-11.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
    "Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>";

  description
    "This YANG module defines an extension statement that allows for
     defining metadata annotations.";

  revision 2014-09-11 {
    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 and may appear only once): 'type', 'description',
         'reference', 'status' and 'units'. If the 'type' statement is
         not present, the built-in 'string' type is used by default.

         A server announces 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).
         Depending on the prescribed usage patterns, the annotation
         then may be attached by the server and/or client to any valid
         instance of a data node defined by the server's data model.

         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 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.

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-00 (work in progress), April 2014.

[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., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-ietf-netconf-restconf-01 (work in progress), July 2014.

Author’s Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 30, 2015

A. Clemm
A. Gonzalez Prieto
E. Voit
Cisco Systems

October 27, 2014

Subscribing to datastore push updates
draft-netmod-clemm-datastore-push-00.txt

Abstract

This document defines a subscription and push mechanism for datastores. This mechanism allows client applications to request updates from a datastore, which are then pushed by the server to the client per a subscription policy, without requiring additional client requests.

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 April 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

YANG datastores, i.e. datastores that contain data modeled according using YANG [RFC6020], are not restricted to configuration data, but can also contain operational data. It is therefore reasonable to expect that data in YANG datastores will increasingly be used to support applications that are not focused on managing configurations but that are, for example, related to service assurance.

Service assurance applications typically involve monitoring operational state of networks and devices; of particular interest are changes that this data undergoes over time. Likewise, there are applications in which data and objects from one datastore need to be made available to applications in other systems and to remote datastores [peermount-req], requiring mechanisms that allow remote

systems to become quickly aware of any updates to allow to validate and maintain cross-network integrity and consistency.

Traditional approaches rely heavily on polling, in which data is periodically explicitly retrieved by a client from a server.

There are various issues associated with polling-based management:

- It introduces additional load on network and devices. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.

- It lacks robustness. Polling cycles may be missed, requests may be delayed or get lost, often particularly in cases when the network is under stress and hence exactly when the need for the data is the greatest.

- Data may be difficult to calibrate and compare. Polling cycles may undergo slight fluctuations, resulting in intervals of different lengths which makes data hard to compare. Likewise, pollers may have difficulty issuing requests that reach all devices at the same time, resulting in offset polling intervals which again make data hard to compare.

More effective is an alternative in which an application can request to be automatically updated of current content of the datastore (such as a subtree, or data in a subtree that meets a certain filter condition), and in which the server subsequently pushes those updates.

The need to perform polling-based management is typically considered an important shortcoming of management applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores using protocols such as NETCONF [RFC6241] or RESTCONF [restconf] will be any more effective, as they would follow the same request/response pattern.

While YANG allows to define notifications, such notifications are generally intended to indicate the occurrence of certain well-specified event conditions, such as a the onset of an alarm condition or the occurrence of an error. Likewise, a capability to define configuration change events has been defined in [RFC5277]. However, these change events pertain only to configuration information, not to operational state. RFC 5277 furthermore predates YANG and does not provide tie-in with YANG-defined datastore contents.

Service Assurance applications are not the only applications benefiting from a push- and subscription-based alternative to polling. Another example is Peer Mount [peermount]. Peer Mount allows a datastore to incorporate data from remote datastores by reference, resulting in virtual datastores that are federated across a network and offer different local views. Various use cases indicate the usefulness of introducing caching in conjunction with Peer Mount, which benefits greatly if updates can automatically be pushed from a mount server to a mount client.

The way in which the updates are to occur can be directed by policy. For example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

Because not every server may support every requested interval for every piece of data, it is furthermore necessary for a server to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate subscription parameters.

Finally, a mechanism is needed to communicate the updates themselves. One option is to use existing NETCONF and RESTCONF mechanisms, by defining special notifications with which to carry those updates. Other alternatives are conceivable, such as use of a dedicated publish/subscribe mechanism that provides an alternative to a NETCONF or RESTCONF transport.

This document specifies a YANG data model for the configuration and management of subscriptions to data in YANG datastores. It also defines a notification that can be used to carry data updates and thus serve as push mechanism.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in a hierarchical manner.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Mount client: The system at which a mount point resides, into which the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.

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.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

Peer Mount: An extension to the YANG management framework that allows local YANG datastores to incorporate data from remote (peer) YANG datastores.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node

SNMP: Simple Network Management Protocol

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Solution Overview

This document specifies a solution that allows clients to subscribe to information updates in a YANG datastore, which are subsequently pushed from the server to the client. The solution encompasses several components:

- The configuration and management of the subscriptions.
- An ability to negotiate subscription parameters where a subscription policy desired by a client cannot be supported.
- The datastream of the push updates.

In addition, there are a number of additional considerations, such as the tie-in of the mechanisms with security mechanisms. Each of those aspects will be discussed in the following subsections.

3.1. Subscription Model

Yang allows modeling the content of notifications. The contents are a set of explicitly stated data nodes forming a hierarchy. For modeling updates in a datastore, a new generic notification is introduced, the "push-update". This notification has the following

semantics. The contents of the notification are not explicitly stated. They are the union of the data nodes in the yang modules supported by the server, excluding the following statements: "mandatory", "must", "min-elements", "max-elements", "when", and "default". Note that the notification contents are dynamic, depending on the modules supported by the server.

Subscriptions to the "push-update" are initiated by clients. Servers respond to a subscription request explicitly positively or negatively. Negative responses include information describing the reason for the subscription rejection.

Datastore-push subscriptions are defined using a data model. This model is based on the subscriptions defined in [RFC-5277], which is also reused in RESTCONF. The model is extended with a subscription type a set of parameters for each type. The complete set of subscription parameters is:

- The name of the stream to subscribe to. The stream is called "push-update".
- The identity of the subscriber.
- An optional filter. It describes the subset of stream events of interest to the subscriber. The server should only send to the subscriber the events that match the filter, when present. The absence of a filter indicates that all events in the stream are of interest to the subscriber and all events in it must be sent to the subscriber. Two filtering mechanisms are considered: subtree filtering and Xpath filtering, with the semantics described in [RFC5277].
- An optional start time. Used to trigger replays starting at the provided time. Its semantics are those in [RFC5277].
- An optional stop time. Used to limit temporarily the events of interest. Its semantics are those in [RFC5277].
- A notification trigger definition. The trigger can be periodic or based on change. For periodic subscriptions, the trigger is defined by the interval with which to push updates. For on-change subscriptions, the trigger is defined using the dampening interval with which to push repeated changes, an indicator forthe magnitude of changes, etc.

The following figure depicts the data model.

```
module: ietf-datastore-push
   +--rw datastore-push-subscription
      +--rw stream             string
      +--rw subscription-id    subscription-identifier
      +--rw (filter)?
      |  +--:(substree)
      |  |  +--rw subtree-filter
      |  +--:(xpath)
      |     +--rw xpath-filter       yang:xpath1.0
      +--rw (notification-trigger)
      |  +--:(periodic)
      |  |  +--rw period             yang:timeticks
      |  +--:(on-change)
      |     +--rw (change-policy)
      |        +--:(delta-policy)
      |           +--rw delta              uint32
      +--rw start-time?        yang:date-and-time
      +--rw stop-time?         yang:date-and-time
```

Figure 1: Model structure

The example below illustrates a subcription for a periodic push of all data under a container called foo.

```
<netconf:rpc message-id="101"
         xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <create-subscription
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
        <stream>push-update</stream>
        <subscription-id>foo</subscription-id>
        <filter netconf:type="xpath"
                  xmlns:ex="http://example.com/dspush/1.0"
                  select="/ex:foo"/>
            <period>500</period>
        </filter>
    </create-subscription>
</netconf:rpc>
```

Figure 2: Subscription example

3.2. Negotiation of Subscription Policies

A subscription rejection can be caused by the inability of the server to provide a stream with the requested semantics. Providing "on-change" updates for operational data can be computationally expensive and an agent may decide not to support them or supporting them for a small number of subscribers or for a limited set of data nodes.

Datastore-push supports a simple negotiation between clients and servers for subscription parameters. The negotiation is limited to a single pair of subscription request and response. For negative responses, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters are no guarantee for subsequent requests for this client or others.

The example below illustrates a subcription response, where an agent does not support frequent periodic updates, and suggests a different sampling rate to the client.

```
<netconf:rpc message-id="101"
         xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <create-subscription
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
        <stream>push-update</stream>
        <subscription-id>foo</subscription-id>
        <filter netconf:type="xpath"
                  xmlns:ex="http://example.com/dspush/1.0"
                  select="/ex:foo"/>
            <period>500</period>
        </filter>
    </create-subscription>
</netconf:rpc>


<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
        <error-type>application</error-type>
        <error-tag>operation-not-supported</error-tag>
        <error-severity>error</error-severity>
        <error-info>
            <supported-subcription>
                <period>3000</period>
            </supported-subcription>
        </error-info>
    </rpc-error>
</rpc-reply>
```

Figure 3: Subscription negotiation example

3.3. Push Data Stream and Transport Mapping

Pushing data based on a subscription could be considered analogous to a response to a data retrieval request, e.g. a "get" request. However, contrary to such a request, multiple responses to the same request may get sent over a longer period of time. Likewise, clients

need to be able to distinguish between data updates and state update regarding the subscription itself, for example when a subscription can no longer be serviced.

A more suitable mechanism is therefore that of a notification. Contrary to notifications associated with alarms and unexpected event occurrences, push updates are solicited, i.e. tied tied to a particular subscription which triggered the notification. (An alternative conceptual model would consider a subscription an "opt-in" filter on a continuous stream of updates.)

The notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.

- A data node that contains a representation of the datastore subtree containing the updates. The subtree is filtered per access control rules to contain only data that the subscriber is authorized to see. Also, depending on the subscription type, i.e., specifically for on-change subscriptions, the subtree contains only the data nodes that contain actual changes. (This can be simply a node of type string or, for XML-based encoding, anyxml.)

Notifications are sent using <notification> elements as defined in [RFC5277]. Alternative transports are conceivable but outside the scope of this specification.

3.4. Other considerations

3.4.1. Authorization

A client may only receive updates to data that the client has proper authorization for. Normal authorization rules apply. Data that is being pushed therefore needs to be subjected to a filter that applies the corresponding rules, removing any non-authorized data as applicable.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536].

3.4.2. Subscription status and subscription monitoring

It is possible that a server may no longer be able to serve a subscription that had been previously accepted. For example, a server may have run out of resources, or internal errors may have

occurred. When this is the case, a server needs to be able to temporarily suspend the subscription, or even to terminate it.

For this reason, a server SHALL maintain status information for each subscription that indicates the current status of the subscription.

In addition, a server needs to indicate any changes in status to the subscriber through a notification. Specifically, subscribers need to be informed of the following:

- A subscription has been temporarily suspended, including the reason. (See subscription-suspended in the model below.)
- A subscription (that had been suspended earlier) is once again operational. (See subscription-resumed in the model below.)
- A subscription has been abnormally terminated, including the reason. (See subscription-terminated in the model below.)

Finally, a server might provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

3.4.3. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, it should be noted that monitoring of operational state changes inside a system can be associated with significant implementation challenges.

Even periodic retrieval of operational state alone, to be able to push it, can consume considerable system resources. Configuration data may in many cases be persisted in an actual database or a configuration file, where retrieval of the database content or the file itself is reasonably straightforward and computationally inexpensive. However, retrieval of operational data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, if is important for an implementation to understand what subscriptions it can or cannot support. It is far preferrable to decline a subscription request, than to accept it only to result in subsequent failure later.

Whether or not a subscription can be supported will in general be determined by a combination of several factors, including the

subscription policy (on-change or periodic, with on-change in general being the more challenging of the two), the period in which to report changes (1 second periods will consume more resources than 1 hour periods), the amount of data in the subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

4. YANG module

<CODE BEGINS>
file "ietf-datastore-push@2014-10-27.yang"

```
 module ietf-datastore-push {
     // RFC Ed.: replace XXXX with 'ietf' and remove this note
     namespace "urn:XXXX:params:xml:ns:yang:ietf-datastore-push";
     prefix "datastore-push";

     import ietf-yang-types { prefix yang; }

     organization
       "IETF";

     contact
       "Editor:   Alexander Clemm
                  <mailto:alex@cisco.com>

        Editor:   Alberto Gonzalez Prieto
                  <mailto:albertgo@cisco.com>

        Editor:   Eric Voit
                  <mailto:evoit@cisco.com>";

     description
       "This module contains conceptual YANG specifications
        for datastore push.";

     revision 2014-10-27 {
       description
         "Initial revision.";
       reference
         "Datastore push.";
     }


    // Typedefs
    typedef datastore-contents {
        type string;
        description
```

```
          "The encoding of the contents adheres to the subscription
           parameters. It corresponds to the filtered datastore
           subtree.";
      }

      typedef subscription-identifier {
          type string {
              length "1 .. max";
          }
          description
            "A client-provided identifier for the subscription.";
      }


      // Identities
      // Subscription error
      identity subscription-errors {
          description
            "Base identity for subscription errors.";
      }

      typedef subscription-term-reason {
          type identityref {
              base "subscription-errors";
          }
          description
            "Reason for a server to terminate a subscription.";
      }

      typedef subscription-susp-reason {
          type identityref {
              base "subscription-errors";
          }
          description
            "Reason for a server to suspend a subscription.";
      }

      identity internal-error {
          base "subscription-errors";
          description
            "Subscription failures caused by server internal error.";
      }

      identity no-resources {
          base "subscription-errors";
          description
            "Lack of resources, e.g. CPU, memory, bandwidth";
      }
```

```
    identity other {
        base "subscription-errors";
        description
          "Fallback reason - any other reason";
    }


    // Notifications
    notification push-update {
        description
          "This notification contains an update from a datastore";

        leaf subscription-id {
            type subscription-identifier;
            mandatory true;
            description
              "This references the subscription because of which the
               notification is sent.";
        }

        leaf datastore-contents {
            type datastore-contents;
            description
              "This contains datastore contents
               per the subscription.";
        }
    }

    notification subscription-suspended {
        description
          "This notification indicates a suspension of the
           subscription by the server has occurred.  No further
           datastore updates will be sent until subscription
           resumes.";

        leaf subscription-id {
            type subscription-identifier;
            mandatory true;
            description
              "This references the affected subscription.";
        }

        leaf reason {
            type subscription-susp-reason;
            description
              "Provides a reason for why the subscription was
               suspended.";
        }
```

```
    }

    notification subscription-resumed {
        description
          "This notification indicates that a subscription that had
           previously been suspended has resumed. Datastore updates
           will once again be sent.";
        leaf subscription-id {
            type subscription-identifier;
            mandatory true;
            description
              "This references the affected subscription.";
        }
    }

    notification subscription-terminated {
        description
          "This notification indicates that a subscription has been
           terminated.";

        leaf subscription-id {
            type subscription-identifier;
            mandatory true;
            description
              "This references the affected subscription.";

        }

        leaf reason {
            type subscription-term-reason;
            description
              "Provides a reason for why the subscription was
              terminated.";
        }
    }


    container datastore-push-subscription {
        description
          "Content of a yang-push subscription.";

        leaf stream {
            type string;
            mandatory true;
            description
              "The name of the stream to subscribe to.";
        }
```

```
            leaf subscription-id {
                type subscription-identifier;
                mandatory true;
                description
                  "Identifier to use for this subscription.";
            }
            choice filter {
                description
                  "Subset of stream events of interest.";
                case substree {
                    container subtree-filter {
                        description
                         "Datastore subtree of interest.";
                    }
                }

                case xpath {
                    leaf xpath-filter {
                        type yang:xpath1.0;
                        mandatory true;
                        description
                          "Xpath defining the events of interest.";
                    }
                }
            }


            choice notification-trigger {
                mandatory true;
                description
                  "Defines necessary conditions for sending an event  to
                   the subscriber.";
                case periodic {
                    description
                      "The agent is requested to notify periodically the
                       current values of the datastore or the subset
                       defined by the filter.";
                    leaf period {
                        type yang:timeticks;
                        mandatory true;
                        description
                          "Elapsed time between notifications.";
                    }
                }

                case on-change {
                    description
                      "The agent is requested to notify changes in
```

```
                    values in the datastore or a subset of it defined
                    by a filter.";

                choice change-policy {
                    mandatory true;
                    description
                      "Policy describing necessary conditions for
                      sending an event  to the subscriber.";
                    case delta-policy {
                        leaf delta {
                            type uint32;
                            mandatory true;
                            description
                              "For integer, minimum difference
                               between current and last reports
                               values that can trigger an update.";
                        }
                    }
                }
            }
        }

        leaf start-time {
            type yang:date-and-time;
            description
              "Starting time for replays.";
            reference "RFC 5277, Section 2.1.1";
        }

        leaf stop-time {
            type yang:date-and-time;
            description
              "Time limit for events of interest.";
            reference "RFC 5277, Section 2.1.1";
        }
    }
}
```

<CODE ENDS>

5. Security Considerations

Subscriptions could be used to attempt to overload servers of YANG datastores. For this reason, it is important that the server has the ability to decline a subscription request if it would deplete its resources. In addition, a server needs to be able to suspend an existing subscription when needed. When this occur, the subscription

status is updated accordingly and the clients are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

6. References

6.1. Normative References

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", STD 15, RFC 1157, May 1990.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[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.

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

6.2. Informative References

[peermount] Clemm, A., Medved, J., and E. Voit, "Mounting YANG-defined information from remote datastores", draft-clemm-netmod-mount-02 (work in progress), October 2014.

[peermount-req] Voit, E., Clemm, A., Bansal, S., Tripathy, A., and P. Yellai, "Requirements for Peer Mounting of YANG subtrees from Remote Datastores", draft-voit-netmod-peer-mount-requirements-00 (work in progress), September 2014.

[restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-03, October 2014.

Authors' Addresses

Alexander Clemm
Cisco Systems

EMail: alex@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

EMail: albertgo@cisco.com

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

NETCONF Data Modeling Language Working Group (netmod)
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

A. Tripathy
E. Voit
A. Clemm
Cisco Systems

October 27, 2014

Cloud SLA YANG Model incorporating Peer Mount Semantics
draft-tripathy-cloud-sla-yang-model-00

Abstract

This document defines a YANG data model which supports cloud SLA applications. Two apps currently operating from this model are a cloud based policer application and a distributed-denial-of-service attack redirect application. Other applications are planned as well.

Key to this model is that it includes semantics for mounting objects from remote network element datastores. These semantics are necessary so that applications can operate on objects distributed across multiple devices as if they were part of a single local datastore.

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 April 30, 2015.

Table of Contents

1. Introduction

This document defines a YANG data model RFC 6020 [RFC6020] for two Cloud SLA applications defined in [peermount-req]. This model would reside in a controller or a network element filling the role of a controller. This document does not define the corresponding YANG models for each of the controlled devices.

Interesting in this model is that it also includes semantics for mounting remote datastores from remote network elements as defined as defined in[draft-clemm-mount]. These semantics are necessary so that applications can operate on objects distributed across multiple devices as if they were part of a single local datastore.

The specific information which is mounted from remote systems are RFC 7223 [RFC7223]YANG operational objects. Effectively the datastore exposes a set of statistics from many devices so that Cloud SLA applications don't have to find and acquire them independently. Also interesting in this model is that it exposes a minimal mix of configuration and operational objects needed for the Cloud SLA category of applications.

This minimal mix is key; it limits the domain of objects exposed and therefore maximizes simplicity from the application developer's viewpoint. This model is being contributed to serve as an

informational guide of how to use [yang-mount]. Implementations of this model currently exist in OpenDaylight.

2. YANG Model

This section provides an overview of the information needed for Cloud SLA services models. Within the section, selected examples are highlighted and corresponding design choices are explained.

2.1. Tree diagram syntax

A simplified graphical representation of the data model is presented in Section 2. The meaning if the symbols in these diagrams are as follows:

- Brackets "[" and "]" enclose list keys.
- Symbols after data node names: "?" means an optional node, and "*" denotes a list and leaf-list.
- Abbreviations before data node names: "rw" means configuration data (read-write), "ro" means state data (read-only), and "M" means mount from a remote datastore.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2.2. Tree diagram structure for the data model

```
+--rw ietf-cloud-sla
   +--rw policies
      +--rw policy [policy-name]
         +--rw policy-name                      string
         +--rw policy-max-bw?                   uint64
         +--ro network-aggregate-bw?            uint64
         +--rw nes
         |  +--rw ne [ne-id]
         |     +--rw ne-id                      string
         |     +--rw policing-policies
         |     |   +--rw policed-bandwidth*     [ifref]
         |     |       +--rw ifref              mounted-interface-ref
         |     |       +--rw bandwidth?         uint64
         |     +--ro interfaces-state
         |     |   +--M interface-statistics
         |     +--rw ddos-prop-conf
         |        +--rw access-list-name?       string
         +--rw ddos
            +--rw ddos?                         empty
            +--ro potential-ddos-underway?      boolean
            +--ro redirect-activated?           boolean
            +--ro redirect-actvation-time?      uint64
```

Key here is the application of the[draft-clemm-mount] syntax:

```
      +--M interface-statistics
```

As formally described in Section 6, this will mount the RFC 7223 [RFC7223]Section 3.2 Target Data Node:

```
      +--ro interfaces-state
```

This Target Data Node contains operational data information from the Interfaces Data Model. In fact it contains information from all interfaces of the network element.

2.3. Of interest in the model

2.3.1. How many objects are exposed by the Mount?

Many fields will be available when mounting this particular Mount Target. Not all will be required for the Cloud SLA applications even if they are available. Some of the ones which are used are identified below. For definitions of these objects, please see RFC 7223 [RFC7223].

```
        +--ro interfaces-state
           +--ro interface* [name]
              +--ro name                    string
              +--ro type                    identityref
              +--ro oper-status             enumeration
              +--ro statistics
                 +--ro in-unicast-pkts?     yang:counter64
                 +--ro in-broadcast-pkts?   yang:counter64
                 +--ro in-multicast-pkts?   yang:counter64
```

A design choice we could have made in the YANG model would be Mount individual interfaces, rather than all interfaces for device. However such a choice would proliferate the number of Mount Bindings required. In the future it is expected that filtering mechanisms will be in place to restrict the number of fields actually available/ accessible across a Mount Binding. Therefore reducing the number of Mount Bindings in a model is a recommended best practice.

2.3.2. Why are Network Element counters exposed under policy?

Counters might be used by other applications, so why would you expose this information so parochially? With data normalization, counters would typically be exposed without having to look at a specific policy. However since we are mounting a read-only copy, normalization is not as critical. Since we are show how information can be mounted locally while maximizing application simplicity.

2.3.3. Interface Ref into Network Element Configuration

The authoritative copy of the individual device policers is located on this controller. It is assumed that the authoritative interface configuration is not. This type of object ownership distribution will be common place.

If we were to directly Mount RFC 7223 [RFC7223] interface info from some remote authoritative source, it would not be possible to add additional objects within this datastore. So rather than leading the datastore tree structure with a list of interfaces, and have the policed bandwidth contained as a data node below the list elements, we instead have a list of policed bandwidth configuration, each referring to an NE/interface that the specific bandwidth configuration pertains. Consistency/integrity is maintained by having NE/interface reference the corresponding nodes in the mounted information. This leverages the model to employ cross-network consistency validation.

2.3.4. Pub/Sub and Cache

It makes little sense for many services to continually ping for Operational statistics since they can be automatically delivered on a schedule. In this case, it is useful to subscribe to changes once, and have them delivered continually. Since Pub/Sub and Caching are capabilities which are required, but which would actually be embodied underneath the YANG syntax, these topic are not explicitly part of this model. Additional drafts to cover these topics are planned.

3. YANG Module

<CODE BEGINS>

```
file "ietf-cloud-sla@2014-10-21.yang"
module ietf-cloud-sla {
namespace "urn:ietf:params:xml:ns:yang:ietf-cloud-sla";
// replace with IANA namespace if it is ever assigned
prefix csla;
import ietf-inet-types {
    prefix "inet";
}
import mount {
    prefix mnt;
}
import ietf-interfaces {
    prefix iif;
}


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: Tom Nadeau
tnadeau@lucidvision.com

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

Editor: Ambika Prasad Tripathy
ambtripa@cisco.com";

description
description
```

```
"This YANG module defines a generic configuration and operational model
for Could Domain Policer + Distributes-Denial-of-Service attach
detection and mitigation application.";

revision 2014-10-21 {
    description "Initial revision.";
    reference "Cloud SLA YANG Model with Mount Semantics";
}

container ietf-cloud-sla {
    description
     "This container defines the set of objects, properties, and rpc
     needed for Cloud SLA applications.  As this container includes a
     Mount Point, several object definitions will be loaded from other
     sources";

    container policies {
        description
         "Provides policy related configuration and operational objects.
         At some point this draft will need to morph in order to be
         compliant with a netmod WG ACL draft.  Since one had not been
         adopted at the time of this draft, we have not created a robust
         ACL or Policy model in this draft.";

        list policy {
            description
             "A Cloud SLA policy which applies across many network
             elements at once, allowing them to be treated as a
             composite entity.";
            key "policy-name";
            leaf policy-name {
                type string;
                description
                 "Defines unique policy name for a controller or network
                 domain. This policy will be associated with a set of
                 routers, possibly including routers from many
                 vendors.";
            }
            leaf policy-max-bw {
                type uint64;
                description
                 "The maximum policed bandwidth incoming to the set of
                 router interfaces where the policy has been applied.
                 This policy will typically be distributed across
                 multiple routers and/or switches.";
            }
            leaf network-aggregate-bw {
                type uint64;
```

```
                config false;
                description
                 "This operational object should be updated by the
                 application.  Every ?x? seconds this will include the
                 sum of aggregated network bandwidth across the set of
                 tracked interfaces.";
            }
            container nes {
                description
                 "This container defines the configuration and
                 operational objects for all the network elements where
                 the cloud wide policy is being applied.";
                list ne {
                    description
                     "Defines the set of properties associated with each
                     network element involved with a global policy. The
                     property may be config or operational.";
                    key "ne-id";
                    leaf ne-id {
                        type string;
                        description
                         "The network element uniquely identified by IP
                         address. This attribute should be used as the
                         key to access information related to a specific
                         network element.  Yes having the ne-id as an IP
                         address as the key is less than optimal.  But
                         it works to simplify this example model.";
                    }
                container policing-policies {
                    list policed-bandwidth {
                     description
                     "Contains bandwidth limits that are in effect for
                     an interface of the remote network element.  Each
                     interface for which a limit is in effect needs to
                     have a corresponding list entry; the absence of a
                     list entry for an interface implies that no limit
                     needing to be policed is in effect.";
                     key "ifref";
                      leaf ifref {
                         type mounted-interface-ref;
                         description
                        "References an interface of the remote network
                        element.";
                     }
                     leaf bandwidth {
                        type uint64;
                        description
                         "Specifies the bandwidth limit, in bps, that
```

```
                        traffic on this interface needs to be policed
                        to.";
                    }
                }
            }
               container interfaces-state {
                      config false;
                         description
                         "Data nodes for the operational state of all
                         interfaces on a network element.";

                         mnt:mountpoint "interface-statistics" {
                             mnt:target "../../../ne-id";
                             mnt:subtree "/iif:interfaces";
                          }
                     }
                     container ddos-prop-conf {
                         description
                          "This container defines the config
                          attributes needed for the network element to
                          handle DDoS traffic scrubbing operation if
                          DDoS attach detected by application. The
                          current way of detecting DDoS attach is
                          based on interface counters for the subnet
                          used in a QoS policy defined by CISCO
                          routers.";
                         leaf access-list-name {
                             type string;
                             description
                              "Defines the preconfigured ACL which can
                              be applied when DDoS traffic detected by
                              the application for fwd traffic DDoS
                              sniffers.";
                         }
                     }
                 }
            container ddos {
                 leaf ddos {
                     type empty ;
                     description
                      "This leaf enables DDoS scrubbing for the
                      policy.";
                 }
                 leaf potential-ddos-underway {
                     type boolean;
                     config false;
                     description
                      "This leaf flag should be updated when DDoS
```

```
                        scrubbing is on.";
                    }
                    leaf redirect-activated {
                        type boolean;
                        config false;
                        description
                         "This leaf flag should be updated by application
                         if acl based forwarding applied for the subnet
                         traffic.";
                    }
                    leaf redirect-actvation-time {
                        type uint64;
                        config false;
                        description
                         "The time duration during which last DDoS attack
                          appeared.";
                    }
                }
            }
        }
    }
    typedef mounted-interface-ref {
         type leafref {
           path "/ietf-cloud-sla/policies/policy/nes/ne/interface-
           state/interface-statistics/
           iif:interfaces/iif:interface/iif:name";
           description
            "This type represents a reference to an interface list
            element that is mounted from a remoted network element.";
       }
    }
}

<CODE ENDS>
```

Beyond the YANG definition itself, we need to insert the expression for the path identifying the ifname in the mounted list. We it might be possible to do this by moving into a "pattern" substatement of the type component of the definition.

```
  typedef mounted-interface-ref {
      type leafref {
          path ?/ietf-cloud-sla/policies/policy[pn]/nes/ne[ne-
          id]/interface-statistics/ifname?;
              description
              "This type represents a reference to an interface list
               element that is mounted from a remoted network element.";
      }
  }
```

4. IANA Considerations

This document makes no request of IANA.

5. Security Considerations

6. Acknowledgements

7. References

7.1. Normative References

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

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

7.2. Informative References

[draft-clemm-mount] "Mounting YANG-Defined Information from Remote Datastores", 2013, <http://tools.ietf.org/id/draft-clemm-netmod-mount-02.txt>.

[peermount-req] Voit, E., "Requirements for Peer Mounting of YANG subtrees from Remote Datastores", October 2014, <http://www.ietf.org/internet-drafts/draft-voit-netmod-peer-mount-requirements-01.txt>.

Authors' Addresses

Ambika Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alex Clemm
Cisco Systems

Email: alex@cisco.com

NETMOD WG
Internet-Draft
Intended status: Informational
Expires: April 27, 2015

Clyde Wildes
Cisco Systems
Agrahara Kiran Koushik
Brocade Communication Systems
Oct 27, 2014

SYSLOG YANG model
draft-wildes-netmod-syslog-model-05

Abstract

This document describes a data model for Syslog protocol which is used to convey event notification messages.

Table of Contents

1. Introduction

Operating systems, processes and applications generate messages indicating their own status or the occurence 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.

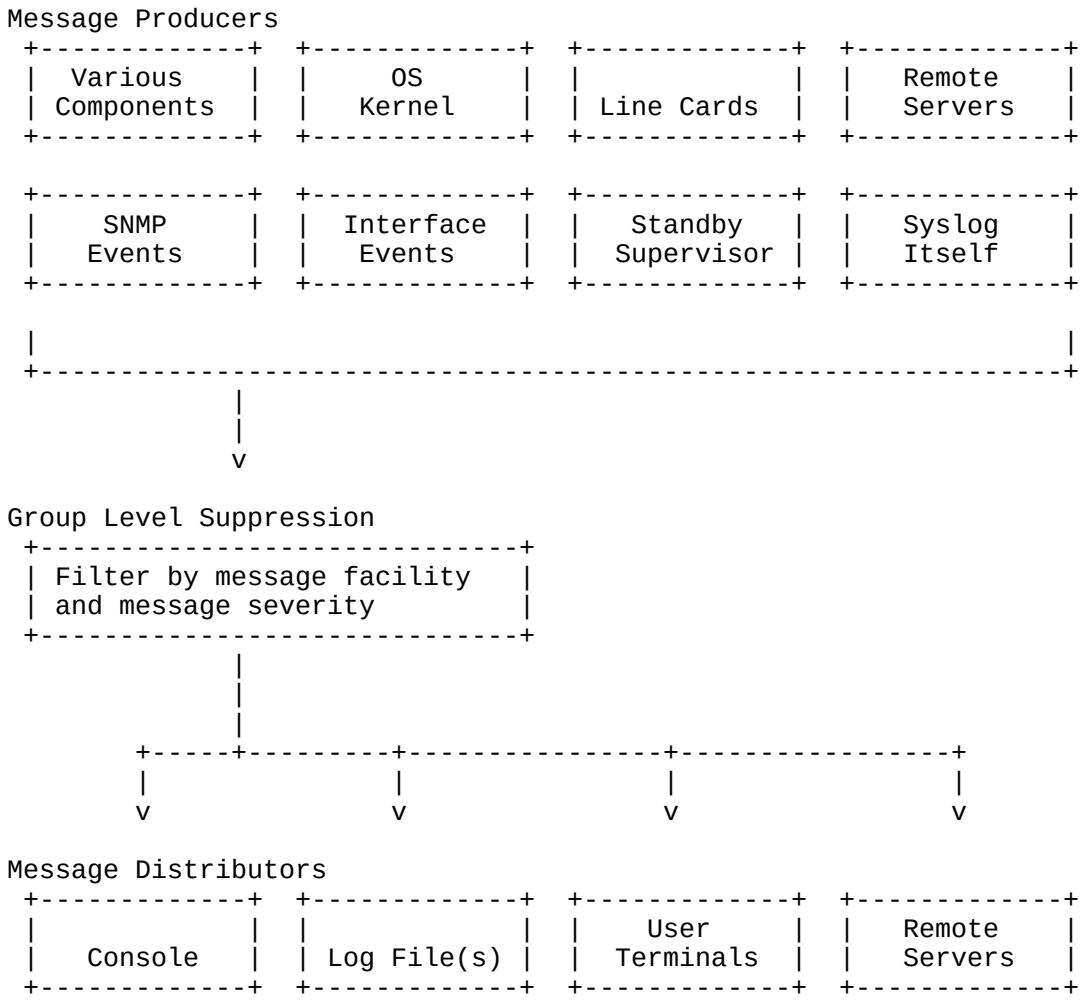
This module makes use of the YANG "feature" construct which allows implementations to support only those Syslog features that lie within their capabilities.

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.



The leaves in the base syslog model correspond to the group level suppression filter and each message distributor:

- console
- 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 (logging-severity-scope)?
      |  |  |     +--:(logging-severity-all)
      |  |  |     |  +--rw all?                                  empty
      |  |  |     +--:(logging-severity)
      |  |  |        +--rw severity?                            syslogtypes:Seve
rity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                                  empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw (logging-severity-scope)?
      |  |           +--:(logging-severity-all)
      |  |           |  +--rw all?        empty
      |  |           +--:(logging-severity)
      |  |              +--rw severity?   syslogtypes:Severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw (logging-severity-operator)?
      |  |     +--:(default)
      |  |     |  +--rw default?      empty
      |  |     +--:(equals)
      |  |     |  +--rw equals?       empty
      |  |     +--:(not-equals)
      |  |        +--rw not-equals?   empty
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |     +--rw pattern-match?   string
      +--rw console-logging-action
      |  +--rw (logging-level-scope)?
      |  |  +--:(logging-facility-all)
      |  |  |  +--rw (logging-severity-scope)?
      |  |  |     +--:(logging-severity-all)
      |  |  |     |  +--rw all?                                  empty
      |  |  |     +--:(logging-severity)
      |  |  |        +--rw severity?                            syslogtypes:Seve
rity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                                  empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw (logging-severity-scope)?
      |  |           +--:(logging-severity-all)
      |  |           |  +--rw all?        empty
      |  |           +--:(logging-severity)
      |  |              +--rw severity?   syslogtypes:Severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw (logging-severity-operator)?
      |  |     +--:(default)
      |  |     |  +--rw default?      empty
      |  |     +--:(equals)
      |  |     |  +--rw equals?       empty
      |  |     +--:(not-equals)
      |  |        +--rw not-equals?   empty
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |     +--rw pattern-match?   string
      +--rw file-logging-action
```

```
      |  +--rw file-name                              inet:uri
      |  +--rw (logging-level-scope)?
      |  |  +--:(logging-facility-all)
      |  |  |  +--rw (logging-severity-scope)?
      |  |  |     +--:(logging-severity-all)
      |  |  |     |  +--rw all?                                 empty
      |  |  |     +--:(logging-severity)
      |  |  |        +--rw severity?                           syslogtypes:Seve
rity
      |  |  +--:(logging-facility-none)
      |  |  |  +--rw none?                                  empty
      |  |  +--:(logging-facility)
      |  |     +--rw logging-facilities* [facility]
      |  |        +--rw facility    identityref
      |  |        +--rw (logging-severity-scope)?
      |  |           +--:(logging-severity-all)
      |  |           |  +--rw all?        empty
      |  |           +--:(logging-severity)
      |  |              +--rw severity?   syslogtypes:Severity
      |  +--rw logging-advanced-level-processing {selector-advanced-level-proces
sing-config}?
      |  |  +--rw (logging-severity-operator)?
      |  |     +--:(default)
      |  |     |  +--rw default?      empty
      |  |     +--:(equals)
      |  |     |  +--rw equals?       empty
      |  |     +--:(not-equals)
      |  |        +--rw not-equals?   empty
      |  +--rw logging-match-processing {selector-match-processing-config}?
      |  |  +--rw pattern-match?   string
      |  +--rw file-logging-structured-data?        boolean {file-logging-struct
ured-data}?
      |  +--rw file-logging-archive {file-logging-archive-config}?
      |     +--rw file-number?       uint32
      |     +--rw file-size?         uint32
      |     +--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 (logging-severity-scope)?
      |     |  |     +--:(logging-severity-all)
      |     |  |     |  +--rw all?                                 empty
      |     |  |     +--:(logging-severity)
      |     |  |        +--rw severity?                           syslogtypes:S
everity
      |     |  +--:(logging-facility-none)
      |     |  |  +--rw none?                                  empty
      |     |  +--:(logging-facility)
      |     |     +--rw logging-facilities* [facility]
      |     |        +--rw facility    identityref
      |     |        +--rw (logging-severity-scope)?
      |     |           +--:(logging-severity-all)
      |     |           |  +--rw all?        empty
      |     |           +--:(logging-severity)
      |     |              +--rw severity?   syslogtypes:Severity
      |     +--rw logging-advanced-level-processing {selector-advanced-level-pro
cessing-config}?
      |     |  +--rw (logging-severity-operator)?
      |     |     +--:(default)
      |     |     |  +--rw default?      empty
      |     |     +--:(equals)
      |     |     |  +--rw equals?       empty
      |     |     +--:(not-equals)
      |     |        +--rw not-equals?   empty
      |     +--rw logging-match-processing {selector-match-processing-config}?
```

```
      |     |  +--rw pattern-match?   string
      |     +--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 certInitialRepeat?   uint16
      |        +--rw certResendDelay?     uint16
      |        +--rw certResendCount?     uint16
      |        +--rw sigMaxDelay?         uint16
      |        +--rw sigNumberResends?    uint16
      |        +--rw sigResendDelay?      uint16
      |        +--rw sigResendCount?      uint16
      +--rw terminal-logging-action
         +--rw (user-scope)?
            +--:(all-users)
            |  +--rw all-users
            |     +--rw (logging-level-scope)?
            |     |  +--:(logging-facility-all)
            |     |  |  +--rw (logging-severity-scope)?
            |     |  |     +--:(logging-severity-all)
            |     |  |     |  +--rw all?                                 empty
            |     |  |     +--:(logging-severity)
            |     |  |        +--rw severity?                           syslogt
ypes:Severity
            |     |  +--:(logging-facility-none)
            |     |  |  +--rw none?                                  empty
            |     |  +--:(logging-facility)
            |     |     +--rw logging-facilities* [facility]
            |     |        +--rw facility    identityref
            |     |        +--rw (logging-severity-scope)?
            |     |           +--:(logging-severity-all)
            |     |           |  +--rw all?        empty
            |     |           +--:(logging-severity)
            |     |              +--rw severity?   syslogtypes:Severity
            |     +--rw logging-advanced-level-processing {selector-advanced-lev
el-processing-config}?
            |     |  +--rw (logging-severity-operator)?
            |     |     +--:(default)
            |     |     |  +--rw default?      empty
            |     |     +--:(equals)
            |     |     |  +--rw equals?       empty
            |     |     +--:(not-equals)
            |     |        +--rw not-equals?   empty
            |     +--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 (logging-severity-scope)?
                  |  |     +--:(logging-severity-all)
                  |  |     |  +--rw all?                                 empty
                  |  |     +--:(logging-severity)
                  |  |        +--rw severity?                           syslogt
ypes:Severity
                  |  +--:(logging-facility-none)
                  |  |  +--rw none?                                  empty
                  |  +--:(logging-facility)
                  |     +--rw logging-facilities* [facility]
                  |        +--rw facility    identityref
                  |        +--rw (logging-severity-scope)?
                  |           +--:(logging-severity-all)
                  |           |  +--rw all?        empty
                  |           +--:(logging-severity)
```

```
                  |              +--rw severity?   syslogtypes:Severity
                  +--rw logging-advanced-level-processing {selector-advanced-lev
el-processing-config}?
                  |  +--rw (logging-severity-operator)?
                  |     +--:(default)
                  |     |  +--rw default?      empty
                  |     +--:(equals)
                  |     |  +--rw equals?       empty
                  |     +--:(not-equals)
                  |        +--rw not-equals?   empty
                  +--rw logging-match-processing {selector-match-processing-conf
ig}?
                     +--rw pattern-match?   string
```

4. SYSLOG YANG Models

4.1. SYSLOG-TYPES module

```
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 2014-10-24 {
    description
      "syslog-model-04 Revision";
    reference
      "This model references RFC 5424 - The Syslog Protocol,
       and RFC 5848 - Signed Syslog Messages.";
  }

  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.";
  }
}
```

4.2. SYSLOG module

```
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: 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 definitions
     for Syslog configuration.";

  revision 2014-10-24 {
    description
      "syslog-model-04 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 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. Choose one of the following:
         logging-severity-all
         logging-severity <severity>";
    choice logging-severity-scope {
      description
        "This choice describes the option to specify all severities
         or a specific severity.";
      case logging-severity-all {
        description
          "This case specifies all severities.";
        leaf all {
          type empty;
          description
          "This leaf specifies that all severities participate in
           the filtering of Syslog messages.";
        }
      }
      case logging-severity {
        description
          "This case specifies a specific severity to participate
           in the filtering of Syslog messages.";
        leaf severity {
          type syslogtypes:Severity;
          description
            "This leaf specifies the Syslog message severity.";
        }
      }
    }
  }

  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 {
      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.";
      choice logging-severity-operator {
        description
          "This choice describes the option to specify how the
           severity comparison is performed.";
        case default {
          description
            "All messages of the specified severity and higher are
             logged according to the given action";
          leaf default {
            type empty;
            description
            "This leaf specifies the default behavior.";
          }
        }
```

```
        case equals {
          description
            "All messages of the specified severity are logged
             according to the given action";
          leaf equals {
            type empty;
            description
            "This leaf specifies all messages for the speicified
             severity.";
          }
        }
        case not-equals {
          description
            "All messages that are not of the specified severity are
             logged according to the given action";
          leaf not-equals {
            type empty;
            description
            "This leaf specifies all messages that are not for the
             speicified severity.";
          }
        }
      }
    }
    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 describes a Posix 1003.2 regular expression
           string that can be used to select a Syslog message for
           logging.";
      }
    }
  }

  container syslog {
    config true;
    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 file-logging-action {
      description
        "This container describes the configuration parameters for
         file logging.";
      leaf file-name {
        type inet:uri;
        mandatory true;
        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;
        description
          "This leaf describes how log messages are written to the
           log file. If set messages will be written in structured-
           data format; if not set 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;
          description
            "This leaf specifies the maximum number of log files
             retained.";
        }
        leaf file-size {
          type uint32;
          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.";
             }
          }
          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;
          mandatory true;
          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 destination-facility {
          type identityref {
            base syslogtypes:syslog-facility;
          }
          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.";
        }
        leaf vrf-name {
          if-feature remote-logging-use-vrf;
          type string;
          description
            "This leaf specifies the name of the virtual routing
             facility (VRF).";
        }
        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 certInitialRepeat {
            type uint16;
            description
            "This leaf specifies the number of times each
             Certificate Block should be sent before the first
             message is sent.";
          }
          leaf certResendDelay {
            type uint16;
            description
              "This leaf specifies the maximum time delay in seconds
               until resending the Certificate Block.";
          }
          leaf certResendCount {
            type uint16;
            description
              "This leaf specifies the maximum number of other
```

```
                syslog messages to send until resending the
                Certificate Block.";
            }
```

```
          leaf sigMaxDelay {
            type uint16;
            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 sigNumberResends {
            type uint16;
            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 sigResendDelay {
            type uint16;
            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 sigResendCount {
            type uint16;
            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 {
        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.";
            }
            uses syslog-selector;
          }
        }
      }
    }
  }
}
```

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>kern</facility><logging-severity>critical</logging-seve
rity>
             </logging-facilities>
             <logging-facilities>
               <facility>auth</facility><logging-severity>error</logging-severit
y>
             </logging-facilities>
           </global-logging-action>
           <console-logging-action>
             <severity>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 NXOS 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]. 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.

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 provided feedback during the writing of this document:

Alexander Clemm <alex@cisco.com>
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, April 2704.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009

[RFC5426] Okmianski, A., "Transmission of Syslog Messages over UDP", RFC 5426, March 2009

[RFC5848] Kelsey, J., Callas, J., Clemm, A., "Signed Syslog Messages", RFC 5848, May 2010.

[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., Bjorklund, M., "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[Posix 1003.2] IEEE, "1003.2-1992 - IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX(R))--Part 2: Shell and Utilities", Posix 1003.2, 1992

Authors' Addresses

Clyde Wildes
Cisco Systems Inc.

Email: cwildes@cisco.com

Kiran Agrahara Sreenivasa
Brocade Communications Systems

Email: kkoushik@brocade.com