

NETMOD WG  
Internet-Draft  
Intended status: Informational  
Expires: December 28, 2014

D. Bogdanovic  
Juniper Networks  
K. Sreenivasa  
Brocade Communications System  
L. Huang  
D. Blair  
Cisco Systems  
June 26, 2014

ACL YANG model  
draft-bogdanovic-netmod-acl-model-01

Abstract

This document describes information and 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 December 28, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Definitions and Acronyms . . . . .	3
2. Problem Statement . . . . .	3
3. Design of the ACL Model . . . . .	3
3.1. ACL Modules . . . . .	4
4. ACL YANG Models . . . . .	6
4.1. IETF-ACL module . . . . .	6
4.2. Packet Header module . . . . .	9
4.3. A company proprietary module example . . . . .	12
4.4. An ACL Example . . . . .	14
5. Linux nftables . . . . .	15
6. Security Considerations . . . . .	15
7. IANA Considerations . . . . .	16
8. Acknowledgements . . . . .	16
9. Change log [RFC Editor: Please remove] . . . . .	16
10. References . . . . .	16
Authors' Addresses . . . . .	17

## 1. Introduction

Access Control List (ACL) are 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 tuple of match criteria and metadata matching.

- o Packet header matches apply to fields visible in the packet such as address or class of service or port numbers.
- o Metadata matching applies 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 can be any sort of thing from counting to

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 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 an access list is. An access list contains an ordered list of rules called access list entries - ACEs. Actions on the first matching ACE are applied with no processing of subsequent 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 matches apply 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 thing from logging to rate limiting or dropping to simply forwarding. There is a default ACE which applies if a packet does not match any of the other ACEs. As some devices fail closed (reject), some fail open (accept) with no explicit configuration, this model defines default action, ACE which applies if a packet does not match any of the other ACEs. This will help with cross platform conformity and for that reason this draft specifies deny as default action. There is overall operational state for the ACL and operational state for each ACE, and depending on the action, operational state for each action. Access-lists can also have notifications such as logging, configuration changing, activation state changes. The ACL can be applied to targets within the device which may be interfaces of a networked device, applications or features running in the device, or other objects. When applied to interfaces of the 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 module "packet-headers" into the match container. The packet headers can be extended with different types and fragments. 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-list
    +--rw acl-name?          string
    +--rw acl-oper-data
      | +--rw match-counter?  ietf:counter64
      | +--rw permit-counter? ietf:counter64
      | +--rw deny-counter?   ietf:counter64
      | +--rw targets*        string
    +--rw access-list-entries* [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-address
| | | | | | | | +--rw flow-label?                    inet:ipv6-flow-lab
| | | | | |
| | | | | +---:(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
| | | | | +--rw ace-oper-data
| | | | | | +--rw match-counter?    ietf:counter64
+--rw default-actions
| | | | | +--rw deny?    empty

```

Two other module "newco-acl" is example of company proprietary model, that augments the ietf-acl module with definitions that are specific to match criteria and company proprietary extensions to match and action criteria. The model below is shown just an example and it is expected from vendors to create their own proprietary models, based on the example below.

```

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

```

#### 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 ietf-acl;

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

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

  revision 2014-05-20{
    description "creating base model for netmod";
  }
}

```

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

container access-list {
  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 {
    description "name of access-list";
    type string;
  }

  container acl-oper-data {
    description "Overall ACL operational data";
    leaf match-counter {
      description "Total match count for ACL";
      type ietf:counter64;
    }
    leaf permit-counter {
      description "Total permit count for ACL";
      type ietf:counter64;
    }
    leaf deny-counter {
      description "Total deny count";
      type ietf:counter64;
    }
    leaf-list targets {
      description "List of targets where ACL is applied";
      type string;
    }
  }

  list access-list-entries {
    key rule-name;
    ordered-by user;
    leaf rule-name {
      description "Entry name";
      type string;
    }
  }
}
```

```
    }

    container matches {
      description "Define match criteria";
      choice ace-type {
        case ace-ip {
          uses packet-fields:acl-ip-header-fields;
          choice ace-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;
        case deny {
          leaf deny {
            type empty;
          }
        }
        case permit {
          leaf permit {
            type empty;
          }
        }
      }
    }

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



```
        container default-actions {
            description "Actions that occur if no access-list entry is matched."
;
            leaf deny {
                type empty;
            }
        }
    }
}
```

#### 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 2014-06-25 {
        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-address;
    }

    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-list/ietf-acl:access-list-entries/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-list/ietf-acl:access-list-entries/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 hierarchical-policer {
            description "Name of hierarchical policer to use to rate-limit traffi
c";
            leaf hierarchitacl-policer{
                type string;
            }
        }
    }
}

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";  
    }  
  }  
}
```

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

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

Figure 2

## 5. Linux nftables

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

## 6. Security Considerations

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

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

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: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

## 8. Acknowledgements

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

## 10. References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC5101] Claise, B., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, January 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.



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

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

Authors' Addresses

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

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: January 2, 2015

G.Galimberti, Ed.  
Cisco  
R.Kunze, Ed.  
Deutsche Telekom  
Kam Lam, Ed.  
Alcatel-Lucent  
D. Hiremagalur, Ed.  
G.Grammel, Ed.  
Juniper  
July 2014

A YANG model to manage t optical interface parameters of DWDM  
applications  
draft-dharini-netmod-g-698-2-yang-00

## Abstract

This memo defines a yang model for managing Optical parameters associated with Dense Wavelength Division Multiplexing (DWDM) interfaces. This is to support the optical parameters described in ITU-T G.698.2. [ITU.G698.2]

The Yang model defined in this memo can be used for Optical Parameters monitoring and/or configuration of the endpoints of Black Links.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

## 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 January 2, 2015.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. The Internet-Standard Management Framework . . . . .	3
3. Conventions . . . . .	3
4. Overview . . . . .	4
4.1. Optical Parameters Description . . . . .	5
4.1.1. Rs-Ss Configuration . . . . .	5
4.1.2. Table of Application Codes . . . . .	6
4.1.3. Table of Vendor Application Codes . . . . .	6
4.2. Optical Interface for G.698.2 . . . . .	7
5. Structure of the Yang Module . . . . .	7
6. Yang Module . . . . .	8
7. Security Considerations . . . . .	12
8. IANA Considerations . . . . .	13
9. Contributors . . . . .	13
10. References . . . . .	14
10.1. Normative References . . . . .	14
10.2. Informative References . . . . .	16
Appendix A. Change Log . . . . .	16
Appendix B. Open Issues . . . . .	16
Authors' Addresses . . . . .	16

## 1. Introduction

This memo defines objects for managing Optical parameters associated with Wavelength Division Multiplexing (WDM) systems in accordance with the optical interface defined in G.698.2 [ITU.G698.2]

Black Link approach allows supporting an optical transmitter/receiver pair of one vendor to inject a DWDM channel and run it over an optical network composed of amplifiers, filters, add-drop multiplexers from a different vendor. From architectural point of

view, the "Black Link" is a set of pre-configured/qualified network connections between the G.698.2 reference points S and R. The black links will be managed at the edges (i.e. the transmitters and receivers attached to the S and R reference points respectively) for the relevant parameters specified in G.698.2 [ITU.G698.2], G.798 [ITU.G798], G.874 [ITU.G874], and the performance parameters specified G.7710/Y.1701 [ITU-T G.7710] and G.874.1 [ITU.G874.1].

The G.698.2 [ITU.G698.2] provides optical parameter values for physical layer interfaces of Dense Wavelength Division Multiplexing (DWDM) systems primarily intended for metro applications which include optical amplifiers. Applications are defined in G.698.2 [ITU.G698.2] using optical interface parameters at the single-channel connection points between optical transmitters and the optical multiplexer, as well as between optical receivers and the optical demultiplexer in the DWDM system. This Recommendation uses a methodology which does not specify the details of the optical link, e.g. the maximum fibre length, explicitly. The Recommendation currently includes unidirectional DWDM applications at 2.5 and 10 Gbit/s (with 100 GHz and 50 GHz channel frequency spacing). Work is still under way for 40 and 100 Gbit/s interfaces. There is possibility for extensions to a lower channel frequency spacing. This document specifically refers to the "application code" defined in the G.698.2 [ITU.G698.2] plus few optical parameter not included in the application code definition.

This draft refers and supports the draft-kunze-g-698-2-management-control-framework

The Yang Model, reporting the Optical parameters and their values, characterizes the features and the performances of the optical components and allow a reliable black link design in case of multi vendor optical networks.

## 2. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

This memo specifies a Yang model for optical interfaces.

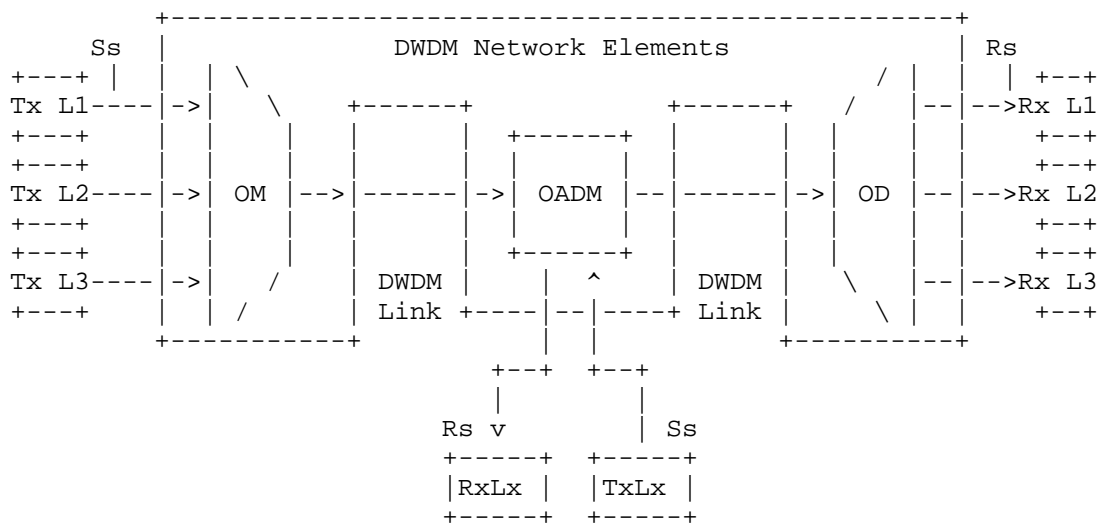
## 3. Conventions

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

the description of OIDs the convention: Set (S) Get (G) and Trap (T) conventions will describe the action allowed by the parameter.

#### 4. Overview

Figure 1 shows a set of reference points, for the linear "black link" approach, for single-channel connection (Ss and Rs) between transmitters (Tx) and receivers (Rx). Here the DWDM network elements include an OM and an OD (which are used as a pair with the opposing element), one or more optical amplifiers and may also include one or more OADMs.



Ss = reference point at the DWDM network element tributary output  
 Rs = reference point at the DWDM network element tributary input  
 Lx = Lambda x  
 OM = Optical Mux  
 OD = Optical Demux  
 OADM = Optical Add Drop Mux

from Fig. 5.1/G.698.2

Figure 1: Linear Black Link

G.698.2 [ITU.G698.2] defines also Ring Black Link configurations [Fig. 5.2/G.698.2] and Bidirectional Black Link configurations [Fig. 5.3/G.698.2]

#### 4.1. Optical Parameters Description

The black links are managed at the edges, i.e. at the transmitters (Tx) and receivers (Rx) attached to the S and R reference points respectively. The parameters that could be managed at the black link edges are specified in G.698.2 [ITU.G698.2] section 5.3 referring the "application code" notation

The definitions of the optical parameters are provided below to increase the readability of the document, where the definition is ended by (G) the parameter can be retrieve with a GET, when (S) it can be provisioned by a SET, (G,S) can be either GET and SET.

##### 4.1.1. Rs-Ss Configuration

The Rs-Ss configuration table allows configuration of Wavelength, Power and Application codes as described in [ITU.G698.2] and G.694.1 [ITU.G694.1]

This parameter report the current Transceiver Output power, it can be either a setting and measured value (G, S).

Wavelength Value (see G.694.1 Table 1):

This parameter indicates the wavelength value that Ss and Rs will be set to work (in THz). See the details in Section 6/G.694.1 (G, S).

Number of Vendor Transceiver Class Supported

This parameter indicates the number of Vendor Transceiver codes supported by this interface (G).

Single-channel application codes(see G.698.2):

This parameter indicates the transceiver application code at Ss and Rs as defined in [ITU.G698.2] Chapter 5.4 - this parameter can be called Optical Interface Identifier OII as per [draft-martinelli-wson-interface-class](G).

Number of Single-channel application codes Supported

This parameter indicates the number of Single-channel application codes supported by this interface (G).

Current Laser Output power:

This parameter report the current Transceiver Output power, it can be either a setting and measured value (G, S).

Current Laser Input power:

This parameter report the current Transceiver Input power (G).

PARAMETERS	Get/Set	Reference
Wavelength Value	G,S	G.694.1 S.6
Vendor Transceiver Class	G	N.A.
Number of Vendor Transceiver Class Supported	G	N.A.
Single-channel application codes	G	G.698.2 S.5.3
Number of Single-channel application codes Supported	G	N.A.
Current Output Power	G,S	N.A.
Current Input Power	G	N.A.

Table 1: Rs-Ss Configuration

## 4.1.2. Table of Application Codes

This table has a list of Application codes supported by this interface at point R are defined in G.698.2.

Application code Identifier:

The Identifier for the Application code.

Application code:

This is the application code that is defined in G.698.2.

## 4.1.3. Table of Vendor Application Codes

This table has a list of Application codes supported by this interface at point R are defined in G.698.2.

Vendor Transceiver Class Identifier::

The Identifier for the vendor transceiver class.

Vendor Transceiver Class:

Other than specifying all the Transceiver parameter, it might be convenient for the vendors to summarize a set of parameters in a single proprietary parameter: the Class of transceiver. The Transceiver classification will be based on the Vendor Name and the main TX and RX parameters (i.e. Trunk Mode, Framing, Bit rate, Trunk Type, Channel Band, Channel Grid, Modulation Format, Channel Modulation Format, FEC Coding, Electrical Signal Framing at Tx, Minimum maximum Chromatic Dispersion (CD) at Rx, Maximum Polarization Mode Dispersion (PMD) at Rx, Maximum differential

group delay at Rx, Loopbacks, TDC, Pre-FEC BER, Q-factor, Q-margin, etc.). If this parameter is used, the vendor will be responsible to specify the Class contents and values. The Vendor can publish the parameters of its Classes or declare to be compatible with published Classes. (G) Optional for compliance. (not mentioned in G.698.2)

#### 4.2. Optical Interface for G.698.2

The ietf-opt-if-g698-2 is an augment to the ietf-interface. It allows the user to set the application code/vendor transceiver class/wavelength and the output power. The module can also be used to get the list of supported application codes/ transceiver class and also the wavelength/output power/input power of the interface.

```

module: ietf-opt-if-g698-2
augment /ietf-interfaces
+.... rw optIfOChRsSs
+... rw ifCurrentApplicationCode
|   + .... rw applicationCodeId
|   + .... rw applicationCode
+... rw ifCurrentVendorTransceiverClass
|   + .... rw vendorTransceiverClassId
|   + .... rw vendorTransceiverClass
+... r  ifSupportedApplicationCodes
|   + .... r  numberApplicationCodesSupported
|   + .... r  optIfOChApplicationCodesList * [application
|                                               CodeId]
|   |   + .... r  applicationCodeId
|   |   + .... r  applicationCode
+... r  ifSupportedVendorTransceiverClasses
|   + .... r  numberTransceiverClassesSupported
|   + .... r  optIfOChVendorTransceiverClassList * [vendor
|                                               TransceiverClassId]
|   |   + .... r  VendorTransceiverClassId
|   |   + .... r  VendorTransceiverClass
+... rw wavelengthn
+... rw outputPower
+... r  InputPower

```

#### 5. Structure of the Yang Module

ietf-opt-if-g698-2 is a top level model for the support of this feature.



## 6. Yang Module

The ietf-opt-if-g698-2 is defined as an extension to ietf interfaces.

```
module ietf-opt-if-g698-2 {
  namespace "urn:ietf:params:xml:ns:yang:ietf-opt-if";
  prefix ietf-opt-if-g698-2

  import ietf-interfaces {
    prefix if;
  }

  import ietf-yang-types {
    prefix yang;
  }
  organization
    "IETF NETMOD (NETCONF Data Modelling 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:     Dharini Hiremagalur
                <mailto:dharinih@juniper.net>";

  description
    "This module contains a collection of YANG definitions for
    configuring Optical interfaces.

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

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

    revision 2014-05-10 {
```

```
        description
            "Initial revision.";
        reference
            "RFC XXXX: A YANG Data Model for Optical Management of
            an Interface for g.698.2 support";
    }

    grouping optIfOCh-VendorTransceiverClass {
        description " A unique Vendor transceiver class supported by
            this interface";
        leaf vendorTransceiverClassId {
            description
                "Id for the Vendor transceiver class";
            type uint8 {
                range "1..255";
            }
        }
        leaf vendorTransceiverClass {
            type string {
                length "1..256";
            }
            description "This defines the transceiver class that
                is/should be used by this interface.
                Vendors can summarize a set of parameters in a
                single proprietary parameter: the Class of
                transceiver. The Transceiver classification will
                be based on the Vendor Name and the main TX and RX
                parameters i.e. Trunk Mode, Framing, Bit rate,
                Trunk Type etc).";
        }
    }
}

grouping optIfOChVendorTransceiverClassesList {
    leaf numberVendorClassesSupported {
        type uint32;
        description " Number of Vendor classes supported by this
            interface";
    }
    list vendorTransceiverList {
        key "vendorTransceiverId";
        uses optIfOCh-VendorTransceiverClass;
    }
}
}
```

```
grouping optIfOChApplicationCode {
    leaf applicationCodeId {
        description
            " Id for the Application code";
        type uint8 {
            range "1..255";
        }
    }
    leaf applicationCode {
        type string {
            length "1..256";
        }
        description "This parameter indicates the transceiver
            application code at Ss and Rs as defined in
            [ITU.G698.2] Chapter 5.3, that is/should be used by
            this interface.
            The optIfOChApplicationsCodeList has all the
            application codes supported by this interface. ";
    }
}

grouping optIfOChApplicationCodesList {
    leaf numberApplicationCodesSupported {
        type uint32;
        description " Number of Application codes supported by
            this interface";
    }
    list applicationCodeList {
        key "applicationCodeId";
        uses optIfApplicationCode;
    }
}

grouping optIfOChOutputPower {
    leaf outputPower {
        type int32;
        units ".01dbm"
        description " The output power for this interface in .01
            dbm "
    }
}

grouping optIfOChInputPower {
    description " Input power of this interface ";
    leaf inputPower {
        type int32;
        units ".01dbm";
    }
}
```

```
        description " The current input power of this interface";
    }
}

grouping optIfOChWavelength {
    leaf wavelengthn {
        type uint32;
        description " This parameter indicate minimum wavelength
        spectrum - n, in a definite wavelength Band (L, C and S)
        as represented in[RFC6205] by the formula -
        Wavelength (nm ) = 1471nm + n* Channel Spacing
                                (converted to nm)
        Eg - Channel Spacing in nm
        'Wavelength (nm ) = 1471nm + n* 20nm (20nm is the
        spacing for CWDM)' ";
    }
}

notification optIfOChWavelengthChange {
    leaf "if-name" {
        type leafref {
            path "/interface/name";
        }
    }
    container wavelength {
        uses optIfOChWavelength;
    }
}

notification optIfOChApplicationCodeChange {
    leaf "if-name" {
        type leafref {
            path "/interface/name";
        }
    }
    container newApplicationCode {
        uses optIfApplicationCode;
    }
}

notification optIfOChVendorTransceiverCodeChange {
    leaf "if-name" {
        type leafref {
            path "/interface/name";
        }
    }
    container newVendorTransceiverClass {
```

```
        uses optIfVendorTransceiverClass;
    }
}

augment "/ietf-interfaces" {

    container optIfOChRsSs {
        description " RsSs path configuration for an interface";

        container ifCurrentApplicationCode {
            uses optIfOChApplicationCode;
        }

        container ifCurrentVendorTransceiverClass {
            uses optIfOChVendorTransceiverClasses;
        }

        container ifSupportedApplicationCodes {
            uses optIfOChApplicationCodesList;
        }

        container ifSupportedVendorTransceiverClasses {
            uses optIfOChVendorTransceiverClassesList;
        }

        uses optIfOChOutputPower;

        uses optIfOChWavelength;

        uses optIfOChInputPower;
    }
}
}
```

## 7. 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 operation and content.

## 8. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:ietf-interfaces:ietf-opt-if-g698-2

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].

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

prefix: ietf-opt-if-g698-2 reference: RFC XXXX

## 9. Contributors

Dean Bogdanovic  
Juniper Networks  
Wesford  
U.S.A.  
email deanb@juniper.net

Arnold Mattheus  
Deutsche Telekom  
Darmstadt  
Germany  
email a.mattheus@telekom.de

Manuel Paul  
Deutsche Telekom  
Berlin  
Germany  
email Manuel.Paul@telekom.de

Walid Wakim  
Cisco  
9501 Technology Blvd  
ROSEMONT, ILLINOIS 60018  
UNITED STATES  
email wwakim@cisco.com

## 10. References

### 10.1. Normative References

- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.
- [RFC3591] Lam, H-K., Stewart, M., and A. Huynh, "Definitions of Managed Objects for the Optical Interface Type", RFC 3591, September 2003.
- [RFC6205] Otani, T. and D. Li, "Generalized Labels for Lambda-Switch-Capable (LSC) Label Switching Routers", RFC 6205, March 2011.
- [ITU.G698.2] International Telecommunications Union, "Amplified multichannel dense wavelength division multiplexing applications with single channel optical interfaces", ITU-T Recommendation G.698.2, November 2009.
- [ITU.G709] International Telecommunications Union, "Interface for the Optical Transport Network (OTN)", ITU-T Recommendation G.709, March 2003.
- [ITU.G872] International Telecommunications Union, "Architecture of optical transport networks", ITU-T Recommendation G.872, November 2001.

- [ITU.G798]  
International Telecommunications Union, "Characteristics of optical transport network hierarchy equipment functional blocks", ITU-T Recommendation G.798, October 2010.
- [ITU.G874]  
International Telecommunications Union, "Management aspects of optical transport network elements", ITU-T Recommendation G.874, July 2010.
- [ITU.G874.1]  
International Telecommunications Union, "Optical transport network (OTN): Protocol-neutral management information model for the network element view", ITU-T Recommendation G.874.1, January 2002.
- [ITU.G959.1]  
International Telecommunications Union, "Optical transport network physical layer interfaces", ITU-T Recommendation G.959.1, November 2009.
- [ITU.G826]  
International Telecommunications Union, "End-to-end error performance parameters and objectives for international, constant bit-rate digital paths and connections", ITU-T Recommendation G.826, November 2009.
- [ITU.G8201]  
International Telecommunications Union, "Error performance parameters and objectives for multi-operator international paths within the Optical Transport Network (OTN)", ITU-T Recommendation G.8201, April 2011.
- [ITU.G694.1]  
International Telecommunications Union, "Spectral grids for WDM applications: DWDM frequency grid", ITU-T Recommendation G.694.1, June 2002.
- [ITU.G7710]  
International Telecommunications Union, "Common equipment management function requirements", ITU-T Recommendation G.7710, May 2008.



## 10.2. Informative References

- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart,  
"Introduction and Applicability Statements for Internet-  
Standard Management Framework", RFC 3410, December 2002.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629,  
June 1999.
- [RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB  
Documents", BCP 111, RFC 4181, September 2005.
- [I-D.kunze-g-698-2-management-control-framework]  
Kunze, R., "A framework for Management and Control of  
optical interfaces supporting G.698.2", draft-kunze-  
g-698-2-management-control-framework-00 (work in  
progress), July 2011.
- [RFC4054] Strand, J. and A. Chiu, "Impairments and Other Constraints  
on Optical Layer Routing", RFC 4054, May 2005.

## Appendix A. Change Log

This optional section should be removed before the internet draft is submitted to the IESG for publication as an RFC.

Note to RFC Editor: please remove this appendix before publication as an RFC.

## Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

## Authors' Addresses

Gabriele Galimberti (editor)  
Cisco  
Via Philips,12  
20052 - Monza  
Italy

Phone: +390392091462  
Email: ggalimbe@cisco.com

Ruediger Kunze (editor)  
Deutsche Telekom  
Dddd, xx  
Berlin  
Germany

Phone: +49xxxxxxxxxx  
Email: RKunze@telekom.de

Hing-Kam Lam (editor)  
Alcatel-Lucent  
600-700 Mountain Avenue, Murray Hill  
New Jersey, 07974  
USA

Phone: +17323313476  
Email: kam.lam@alcatel-lucent.com

Dharini Hiremagalur (editor)  
Juniper  
1194 N Mathilda Avenue  
Sunnyvale - 94089 California  
USA

Phone: +1408  
Email: dharinih@juniper.net

Gert Grammel (editor)  
Juniper  
1194 N Mathilda Avenue  
Sunnyvale - 94089 California  
USA

Phone: +1408  
Email: ggrammel@juniper.net

Network Working Group  
Internet-Draft  
Obsoletes: 6020 (if approved)  
Intended status: Standards Track  
Expires: January 4, 2015

M. Bjorklund, Ed.  
Tail-f Systems  
July 3, 2014

YANG - A Data Modeling Language for the Network Configuration Protocol  
(NETCONF)  
draft-ietf-netmod-rfc6020bis-00

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 January 4, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	8
1.1. Summary of Changes from RFC 6020 . . . . .	8
2. Keywords . . . . .	8
3. Terminology . . . . .	8
3.1. Mandatory Nodes . . . . .	11
4. YANG Overview . . . . .	11
4.1. Functional Overview . . . . .	11
4.2. Language Overview . . . . .	13
4.2.1. Modules and Submodules . . . . .	13
4.2.2. Data Modeling Basics . . . . .	13
4.2.3. State Data . . . . .	18
4.2.4. Built-In Types . . . . .	18
4.2.5. Derived Types (typedef) . . . . .	19
4.2.6. Reusable Node Groups (grouping) . . . . .	20
4.2.7. Choices . . . . .	21
4.2.8. Extending Data Models (augment) . . . . .	22
4.2.9. RPC Definitions . . . . .	23
4.2.10. Notification Definitions . . . . .	24
5. Language Concepts . . . . .	25
5.1. Modules and Submodules . . . . .	25
5.1.1. Import and Include by Revision . . . . .	26
5.1.2. Module Hierarchies . . . . .	27
5.2. File Layout . . . . .	28
5.3. XML Namespaces . . . . .	28
5.3.1. YANG XML Namespace . . . . .	29
5.4. Resolving Grouping, Type, and Identity Names . . . . .	29
5.5. Nested Typedefs and Groupings . . . . .	29
5.6. Conformance . . . . .	30
5.6.1. Basic Behavior . . . . .	30
5.6.2. Optional Features . . . . .	31
5.6.3. Deviations . . . . .	31

5.6.4.	Announcing Conformance Information in the <hello> Message . . . . .	32
5.7.	Data Store Modification . . . . .	34
6.	YANG Syntax . . . . .	34
6.1.	Lexical Tokenization . . . . .	34
6.1.1.	Comments . . . . .	34
6.1.2.	Tokens . . . . .	35
6.1.3.	Quoting . . . . .	35
6.2.	Identifiers . . . . .	36
6.2.1.	Identifiers and Their Namespaces . . . . .	36
6.3.	Statements . . . . .	37
6.3.1.	Language Extensions . . . . .	38
6.4.	XPath Evaluations . . . . .	38
6.4.1.	XPath Context . . . . .	38
6.5.	Schema Node Identifier . . . . .	39
7.	YANG Statements . . . . .	39
7.1.	The module Statement . . . . .	40
7.1.1.	The module's Substatements . . . . .	41
7.1.2.	The yang-version Statement . . . . .	41
7.1.3.	The namespace Statement . . . . .	42
7.1.4.	The prefix Statement . . . . .	42
7.1.5.	The import Statement . . . . .	42
7.1.6.	The include Statement . . . . .	43
7.1.7.	The organization Statement . . . . .	44
7.1.8.	The contact Statement . . . . .	44
7.1.9.	The revision Statement . . . . .	44
7.1.10.	Usage Example . . . . .	45
7.2.	The submodule Statement . . . . .	46
7.2.1.	The submodule's Substatements . . . . .	47
7.2.2.	The belongs-to Statement . . . . .	48
7.2.3.	Usage Example . . . . .	49
7.3.	The typedef Statement . . . . .	49
7.3.1.	The typedef's Substatements . . . . .	50
7.3.2.	The typedef's type Statement . . . . .	50
7.3.3.	The units Statement . . . . .	50
7.3.4.	The typedef's default Statement . . . . .	50
7.3.5.	Usage Example . . . . .	51
7.4.	The type Statement . . . . .	51
7.4.1.	The type's Substatements . . . . .	51
7.5.	The container Statement . . . . .	51
7.5.1.	Containers with Presence . . . . .	52
7.5.2.	The container's Substatements . . . . .	52
7.5.3.	The must Statement . . . . .	53
7.5.4.	The must's Substatements . . . . .	54
7.5.5.	The presence Statement . . . . .	55
7.5.6.	The container's Child Node Statements . . . . .	56
7.5.7.	XML Mapping Rules . . . . .	56
7.5.8.	NETCONF <edit-config> Operations . . . . .	56

7.5.9.	Usage Example . . . . .	57
7.6.	The leaf Statement . . . . .	58
7.6.1.	The leaf's default value . . . . .	58
7.6.2.	The leaf's Substatements . . . . .	59
7.6.3.	The leaf's type Statement . . . . .	59
7.6.4.	The leaf's default Statement . . . . .	59
7.6.5.	The leaf's mandatory Statement . . . . .	59
7.6.6.	XML Mapping Rules . . . . .	60
7.6.7.	NETCONF <edit-config> Operations . . . . .	60
7.6.8.	Usage Example . . . . .	60
7.7.	The leaf-list Statement . . . . .	61
7.7.1.	Ordering . . . . .	62
7.7.2.	The leaf-list's Substatements . . . . .	62
7.7.3.	The min-elements Statement . . . . .	63
7.7.4.	The max-elements Statement . . . . .	63
7.7.5.	The ordered-by Statement . . . . .	64
7.7.6.	XML Mapping Rules . . . . .	64
7.7.7.	NETCONF <edit-config> Operations . . . . .	65
7.7.8.	Usage Example . . . . .	66
7.8.	The list Statement . . . . .	67
7.8.1.	The list's Substatements . . . . .	67
7.8.2.	The list's key Statement . . . . .	68
7.8.3.	The list's unique Statement . . . . .	69
7.8.4.	The list's Child Node Statements . . . . .	70
7.8.5.	XML Mapping Rules . . . . .	70
7.8.6.	NETCONF <edit-config> Operations . . . . .	71
7.8.7.	Usage Example . . . . .	72
7.9.	The choice Statement . . . . .	75
7.9.1.	The choice's Substatements . . . . .	75
7.9.2.	The choice's case Statement . . . . .	76
7.9.3.	The choice's default Statement . . . . .	77
7.9.4.	The choice's mandatory Statement . . . . .	78
7.9.5.	XML Mapping Rules . . . . .	79
7.9.6.	NETCONF <edit-config> Operations . . . . .	79
7.9.7.	Usage Example . . . . .	79
7.10.	The anyxml Statement . . . . .	80
7.10.1.	The anyxml's Substatements . . . . .	81
7.10.2.	XML Mapping Rules . . . . .	81
7.10.3.	NETCONF <edit-config> Operations . . . . .	81
7.10.4.	Usage Example . . . . .	82
7.11.	The grouping Statement . . . . .	82
7.11.1.	The grouping's Substatements . . . . .	83
7.11.2.	Usage Example . . . . .	83
7.12.	The uses Statement . . . . .	84
7.12.1.	The uses's Substatements . . . . .	84
7.12.2.	The refine Statement . . . . .	84
7.12.3.	XML Mapping Rules . . . . .	85
7.12.4.	Usage Example . . . . .	85

7.13. The rpc Statement . . . . .	87
7.13.1. The rpc's Substatements . . . . .	87
7.13.2. The input Statement . . . . .	87
7.13.3. The output Statement . . . . .	88
7.13.4. XML Mapping Rules . . . . .	89
7.13.5. Usage Example . . . . .	89
7.14. The notification Statement . . . . .	90
7.14.1. The notification's Substatements . . . . .	91
7.14.2. XML Mapping Rules . . . . .	91
7.14.3. Usage Example . . . . .	91
7.15. The augment Statement . . . . .	92
7.15.1. The augment's Substatements . . . . .	93
7.15.2. XML Mapping Rules . . . . .	93
7.15.3. Usage Example . . . . .	93
7.16. The identity Statement . . . . .	95
7.16.1. The identity's Substatements . . . . .	95
7.16.2. The base Statement . . . . .	96
7.16.3. Usage Example . . . . .	96
7.17. The extension Statement . . . . .	97
7.17.1. The extension's Substatements . . . . .	98
7.17.2. The argument Statement . . . . .	98
7.17.3. Usage Example . . . . .	99
7.18. Conformance-Related Statements . . . . .	99
7.18.1. The feature Statement . . . . .	99
7.18.2. The if-feature Statement . . . . .	101
7.18.3. The deviation Statement . . . . .	101
7.19. Common Statements . . . . .	104
7.19.1. The config Statement . . . . .	104
7.19.2. The status Statement . . . . .	104
7.19.3. The description Statement . . . . .	105
7.19.4. The reference Statement . . . . .	105
7.19.5. The when Statement . . . . .	106
8. Constraints . . . . .	107
8.1. Constraints on Data . . . . .	107
8.2. Hierarchy of Constraints . . . . .	107
8.3. Constraint Enforcement Model . . . . .	108
8.3.1. Payload Parsing . . . . .	108
8.3.2. NETCONF <edit-config> Processing . . . . .	109
8.3.3. Validation . . . . .	109
9. Built-In Types . . . . .	110
9.1. Canonical Representation . . . . .	110
9.2. The Integer Built-In Types . . . . .	111
9.2.1. Lexical Representation . . . . .	111
9.2.2. Canonical Form . . . . .	112
9.2.3. Restrictions . . . . .	112
9.2.4. The range Statement . . . . .	112
9.2.5. Usage Example . . . . .	113
9.3. The decimal64 Built-In Type . . . . .	113

9.3.1.	Lexical Representation . . . . .	113
9.3.2.	Canonical Form . . . . .	114
9.3.3.	Restrictions . . . . .	114
9.3.4.	The fraction-digits Statement . . . . .	114
9.3.5.	Usage Example . . . . .	115
9.4.	The string Built-In Type . . . . .	115
9.4.1.	Lexical Representation . . . . .	115
9.4.2.	Canonical Form . . . . .	115
9.4.3.	Restrictions . . . . .	115
9.4.4.	The length Statement . . . . .	115
9.4.5.	Usage Example . . . . .	116
9.4.6.	The pattern Statement . . . . .	117
9.4.7.	Usage Example . . . . .	117
9.5.	The boolean Built-In Type . . . . .	118
9.5.1.	Lexical Representation . . . . .	118
9.5.2.	Canonical Form . . . . .	118
9.5.3.	Restrictions . . . . .	118
9.6.	The enumeration Built-In Type . . . . .	118
9.6.1.	Lexical Representation . . . . .	118
9.6.2.	Canonical Form . . . . .	118
9.6.3.	Restrictions . . . . .	118
9.6.4.	The enum Statement . . . . .	118
9.6.5.	Usage Example . . . . .	119
9.7.	The bits Built-In Type . . . . .	120
9.7.1.	Restrictions . . . . .	120
9.7.2.	Lexical Representation . . . . .	120
9.7.3.	Canonical Form . . . . .	120
9.7.4.	The bit Statement . . . . .	120
9.7.5.	Usage Example . . . . .	121
9.8.	The binary Built-In Type . . . . .	122
9.8.1.	Restrictions . . . . .	122
9.8.2.	Lexical Representation . . . . .	122
9.8.3.	Canonical Form . . . . .	122
9.9.	The leafref Built-In Type . . . . .	122
9.9.1.	Restrictions . . . . .	122
9.9.2.	The path Statement . . . . .	123
9.9.3.	Lexical Representation . . . . .	123
9.9.4.	Canonical Form . . . . .	124
9.9.5.	Usage Example . . . . .	124
9.10.	The identityref Built-In Type . . . . .	127
9.10.1.	Restrictions . . . . .	127
9.10.2.	The identityref's base Statement . . . . .	127
9.10.3.	Lexical Representation . . . . .	128
9.10.4.	Canonical Form . . . . .	128
9.10.5.	Usage Example . . . . .	128
9.11.	The empty Built-In Type . . . . .	129
9.11.1.	Restrictions . . . . .	129
9.11.2.	Lexical Representation . . . . .	129



9.11.3.	Canonical Form . . . . .	129
9.11.4.	Usage Example . . . . .	129
9.12.	The union Built-In Type . . . . .	130
9.12.1.	Restrictions . . . . .	130
9.12.2.	Lexical Representation . . . . .	130
9.12.3.	Canonical Form . . . . .	131
9.13.	The instance-identifier Built-In Type . . . . .	131
9.13.1.	Restrictions . . . . .	132
9.13.2.	The require-instance Statement . . . . .	132
9.13.3.	Lexical Representation . . . . .	132
9.13.4.	Canonical Form . . . . .	132
9.13.5.	Usage Example . . . . .	132
10.	Updating a Module . . . . .	133
11.	YIN . . . . .	135
11.1.	Formal YIN Definition . . . . .	136
11.1.1.	Usage Example . . . . .	138
12.	YANG ABNF Grammar . . . . .	139
13.	Error Responses for YANG Related Errors . . . . .	161
13.1.	Error Message for Data That Violates a unique Statement	161
13.2.	Error Message for Data That Violates a max-elements Statement . . . . .	162
13.3.	Error Message for Data That Violates a min-elements Statement . . . . .	162
13.4.	Error Message for Data That Violates a must Statement	162
13.5.	Error Message for Data That Violates a require-instance Statement . . . . .	163
13.6.	Error Message for Data That Does Not Match a leafref Type . . . . .	163
13.7.	Error Message for Data That Violates a mandatory choice Statement . . . . .	163
13.8.	Error Message for the "insert" Operation . . . . .	163
14.	IANA Considerations . . . . .	164
14.1.	Media type application/yang . . . . .	165
14.2.	Media type application/yin+xml . . . . .	165
15.	Security Considerations . . . . .	167
16.	Contributors . . . . .	167
17.	Acknowledgements . . . . .	168
18.	ChangeLog . . . . .	168
18.1.	Version -00 . . . . .	168
19.	References . . . . .	168
19.1.	Normative References . . . . .	168
19.2.	Informative References . . . . .	169
	Author's Address . . . . .	170

## 1. Introduction

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

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

### 1.1. Summary of Changes from RFC 6020

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

- o Changed the YANG version from "1" to "1.1".
- o Made the "yang-version" statement mandatory.

## 2. Keywords

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

## 3. Terminology

- o anyxml: A data node that can contain an unknown chunk of XML data.
- o augment: Adds new schema nodes to a previously defined schema node.
- o base type: The type from which a derived type was derived, which may be either a built-in type or another derived type.
- o built-in type: A YANG data type defined in the YANG language, such as uint32 or string.
- o choice: A schema node where only one of a number of identified alternatives is valid.

- o configuration data: The set of writable data that is required to transform a system from its initial default state into its current state [RFC6241].
- o conformance: A measure of how accurately a device follows a data model.
- o container: An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes.
- o data definition statement: A statement that defines new data nodes. One of container, leaf, leaf-list, list, choice, case, augment, uses, and anyxml.
- o data model: A data model describes how data is represented and accessed.
- o data node: A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, and anyxml.
- o data tree: The instantiated tree of configuration and state data on a device.
- o derived type: A type that is derived from a built-in type (such as uint32), or another derived type.
- o device deviation: A failure of the device to implement the module faithfully.
- o extension: An extension attaches non-YANG semantics to statements. The extension statement defines new statements to express these semantics.
- o feature: A mechanism for marking a portion of the model as optional. Definitions can be tagged with a feature name and are only valid on devices that support that feature.
- o grouping: A reusable set of schema nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it. The grouping statement is not a data definition statement and, as such, does not define any nodes in the schema tree.
- o identifier: Used to identify different kinds of YANG items by name.

- o instance identifier: A mechanism for identifying a particular node in a data tree.
- o interior node: Nodes within a hierarchy that are not leaf nodes.
- o leaf: A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.
- o leaf-list: Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.
- o list: An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.
- o module: A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and "compilable".
- o RPC: A Remote Procedure Call, as used within the NETCONF protocol.
- o RPC operation: A specific Remote Procedure Call, as used within the NETCONF protocol. It is also called a protocol operation.
- o schema node: A node in the schema tree. One of container, leaf, leaf-list, list, choice, case, rpc, input, output, notification, and anyxml.
- o schema node identifier: A mechanism for identifying a particular node in the schema tree.
- o schema tree: The definition hierarchy specified within a module.
- o state data: The additional data on a system that is not configuration data such as read-only status information and collected statistics [RFC6241].
- o submodule: A partial module definition that contributes derived types, groupings, data nodes, RPCs, and notifications to a module. A YANG module can be constructed from a number of submodules.
- o top-level data node: A data node where there is no other data node between it and a module or submodule statement.
- o uses: The "uses" statement is used to instantiate the set of schema nodes defined in a grouping statement. The instantiated

nodes may be refined and augmented to tailor them to any specific needs.

### 3.1. Mandatory Nodes

A mandatory node is one of:

- o A leaf, choice, or anyxml node with a "mandatory" statement with the value "true".
- o A list or leaf-list node with a "min-elements" statement with a value greater than zero.
- o A container node without a "presence" statement, which has at least one mandatory node as a child.

## 4. YANG Overview

### 4.1. Functional Overview

YANG is a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, Remote Procedure Calls (RPCs), and notifications. This allows a complete description of all data sent between a NETCONF client and server.

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

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

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

YANG defines a set of built-in types, and has a type mechanism through which additional types may be defined. Derived types can restrict their base type's set of valid values using mechanisms like range or pattern restrictions that can be enforced by clients or

servers. They can also define usage conventions for use of the derived type, such as a string-based type that contains a host name.

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

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

YANG modules can be translated into an equivalent XML syntax called YANG Independent Notation (YIN) (Section 11), 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 an element from one case is created, all elements 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:

- o For a module or submodule to reference definitions in an external module, the external module MUST be imported.
- o For a module to reference definitions in one of its submodules, the module MUST include the submodule.

- o 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.

#### 5.1.1.1. 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 14.

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:

- o the basic behavior of the model
- o optional features that are part of the model
- o 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 with the name of the feature as its argument.

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&
      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&
      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&
      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:

<code>\n</code>	new line
<code>\t</code>	a tab character
<code>\"</code>	a double quote
<code>\\</code>	a single backslash

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 12. Identifiers can be specified as quoted or unquoted strings.

##### 6.2.1. Identifiers and Their Namespaces

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

- o All module and submodule names share the same global module identifier namespace.
- o All extension names defined in a module and its submodules share the same extension identifier namespace.
- o All feature names defined in a module and its submodules share the same feature identifier namespace.
- o All identity names defined in a module and its submodules share the same identity identifier namespace.
- o All derived type names defined within a parent node or at the top level of the module or its submodules share the same type identifier namespace. This namespace is scoped to all descendant nodes of the parent node or module. This means that any descendent node may use that typedef, and it MUST NOT define a typedef with the same name.
- o All grouping names defined within a parent node or at the top level of the module or its submodules share the same grouping identifier namespace. This namespace is scoped to all descendant nodes of the parent node or module. This means that any descendent node may use that grouping, and it MUST NOT define a grouping with the same name.
- o All leafs, leaf-lists, lists, containers, choices, rpcs, notifications, and anyxmls defined (directly or through a uses statement) within a parent node or at the top level of the module or its submodules share the same identifier namespace. This namespace is scoped to the parent node or module, unless the parent node is a case node. In that case, the namespace is scoped to the closest ancestor node that is not a case or choice node.
- o All cases within a choice share the same case identifier namespace. This namespace is scoped to the parent choice node.

Forward references are allowed in YANG.

### 6.3. Statements

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

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

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

#### 6.3.1. Language Extensions

A module can introduce YANG extensions by using the "extension" keyword (see Section 7.17). 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 12), 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:

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



- o Names without a namespace prefix belong to the same namespace as the identifier of the current node. Inside a grouping, that namespace is affected by where the grouping is used (see Section 7.12).
- o The function library is the core function library defined in [XPath], and a function "current()" that returns a node set with the initial context node.
- o The set of variable bindings is empty.

The mechanism for handling unprefixed names is adopted from XPath 2.0 [XPath2.0], and helps simplify XPath expressions in YANG. No ambiguity may ever arise because YANG node identifiers are always qualified names with a non-null namespace URI.

The 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 12, 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 14.

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:

- o use any grouping and typedef defined at the top level in the imported module or its submodules.
- o use any extension, feature, and identity defined in the imported module or its submodules.
- o use any node in the imported module's schema tree in "must", "path", and "when" statements, or as the target node in "augment" and "deviation" statements.

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

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

Multiple revisions of the same module MUST NOT be imported.

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

#### The import's Substatements

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

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

##### 7.1.6. The include Statement

The "include" statement is used to make content from a submodule available to that submodule's parent module, 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 initial "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 14.

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	9.4.6	0..n
range	9.2.4	0..1
require-instance	9.13.2	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:

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

- o 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.
- o 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.
- o 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.
- o 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.
- o 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:

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

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

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

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

## 7.6.2. The leaf's Substatements

substatement	section	cardinality
config	7.19.1	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):

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

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

This constraint is enforced according to the rules in Section 8.

#### 7.6.6. XML Mapping Rules

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

The value of the leaf node is encoded to XML according to the type, and sent as character data in the element.

A NETCONF server that replies to a <get> or <get-config> request MAY choose not to send the leaf element if its value is the default value. Thus, a client that receives an <rpc-reply> for a <get> or <get-config> request, MUST be prepared to handle the case that a leaf node with a default value is not present in the XML. In this case, the value used by the server is known to be the default value.

See Section 7.6.8 for an example.

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

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

If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the value found in the XML RPC data.

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

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

#### 7.6.8. Usage Example

Given the following "leaf" statement, placed in the previously defined "ssh" container (see Section 7.5.9):

```
leaf port {  
    type inet:port-number;  
    default 22;  
    description "The port to which the SSH server listens"  
}
```

A corresponding XML instance example:

```
<port>2022</port>
```

To set the value of a leaf with an <edit-config>:

```
<rpc message-id="101"  
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <edit-config>  
    <target>  
      <running/>  
    </target>  
    <config>  
      <system xmlns="http://example.com/schema/config">  
        <services>  
          <ssh>  
            <port>2022</port>  
          </ssh>  
        </services>  
      </system>  
    </config>  
  </edit-config>  
</rpc>
```

### 7.7. The leaf-list Statement

Where the "leaf" statement is used to define a simple scalar variable of a particular type, the "leaf-list" statement is used to define an array of a particular type. The "leaf-list" statement takes one argument, which is an identifier, followed by a block of substatements that holds detailed leaf-list information.

The values in a leaf-list MUST be unique.

Conceptually, the values in the data tree are always in the canonical form (see Section 9.1).

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):

- o If this ancestor is a case node, the constraint is enforced if any other node from the case exists.
- o Otherwise, it is enforced if the ancestor node exists.

The constraint is further enforced according to the rules in Section 8.

### 7.7.4. 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 12.

#### 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 12). 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 12.

##### 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
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", "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):

- o If this ancestor is a case node, the constraint is enforced if any other node from the case exists.
- o Otherwise, it is enforced if the ancestor node exists.

The constraint is further enforced according to the rules in Section 8.

#### 7.9.5. 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:

- o A leaf or choice node may get a default value, or a new default value if it already had one.
- o Any node may get a specialized "description" string.
- o Any node may get a specialized "reference" string.
- o Any node may get a different "config" statement.
- o A leaf, anyxml, or choice node may get a different "mandatory" statement.
- o A container node may get a "presence" statement.
- o A leaf, leaf-list, list, container, or anyxml node may get additional "must" expressions.
- o A leaf-list or list node may get a different "min-elements" or "max-elements" statement.

#### 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 12) 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 12) 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:

- o It is irreflexive, which means that an identity is not derived from itself.
- o 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 12.

#### 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 11).

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 a feature are ignored by the device unless the device supports the given feature. 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 the name of a feature, as defined by a "feature" statement. The parent statement is implemented by servers that support this feature. If a prefix is present on the feature name, it 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.

## 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".

- o "current" means that the definition is current and valid.
- o "deprecated" indicates an obsolete definition, but it permits new/continued implementation in order to foster interoperability with older/existing implementations.

- o "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.

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:

- o If the "when" statement is a child of an "augment" statement, then the context node is the augment's target node in the data tree, if the target node is a data node. Otherwise, the context node is the closest ancestor node to the target node that is also a data node.
- o If the "when" statement is a child of a "uses", "choice", or "case" statement, then the context node is the closest ancestor node to the "uses", "choice", or "case" node that is also a data node.
- o If the "when" statement is a child of any other data definition statement, the context node is the data definition's node in the data tree.
- o 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:

- o 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.
- o 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.
- o 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.

- o 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.
- o 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.

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

### 8.2. Hierarchy of Constraints

Conditions on parent nodes affect constraints on child nodes as a natural consequence of the hierarchy of nodes. "must", "mandatory", "min-elements", and "max-elements" constraints are not enforced if

the parent node has a "when" or "if-feature" property that is not satisfied on the current device.

In this example, the "mandatory" constraint on the "longitude" leaf is not enforced on devices that lack the "has-gps" feature:

```
    container location {
      if-feature has-gps;
      leaf longitude {
        mandatory true;
        ...
      }
    }
```

### 8.3. Constraint Enforcement Model

For configuration data, there are three windows when constraints MUST be enforced:

- o during parsing of RPC payloads
- o during processing of NETCONF operations
- o during validation

Each of these scenarios is considered in the following sections.

#### 8.3.1. Payload Parsing

When content arrives in RPC payloads, it MUST be well-formed XML, following the hierarchy and content rules defined by the set of models the device implements.

- o If a leaf data value does not match the type constraints for the leaf, including those defined in the type's "range", "length", and "pattern" properties, the server MUST reply with an "invalid-value" error-tag in the rpc-error, and with the error-app-tag and error-message associated with the constraint, if any exist.
- o If all keys of a list entry are not present, the server MUST reply with a "missing-element" error-tag in the rpc-error.
- o If data for more than one case branch of a choice is present, the server MUST reply with a "bad-element" in the rpc-error.

- o If data for a node tagged with "if-feature" is present, and the feature is not supported by the device, the server MUST reply with an "unknown-element" error-tag in the rpc-error.
- o If data for a node tagged with "when" is present, and the "when" condition evaluates to "false", the server MUST reply with an "unknown-element" error-tag in the rpc-error.
- o For insert handling, if the value for the attributes "before" and "after" are not valid for the type of the appropriate key leafs, the server MUST reply with a "bad-attribute" error-tag in the rpc-error.
- o If the attributes "before" and "after" appears in any element that is not a list whose "ordered-by" property is "user", the server MUST reply with an "unknown-attribute" error-tag in the rpc-error.

#### 8.3.2. NETCONF <edit-config> Processing

After the incoming data is parsed, the NETCONF server performs the <edit-config> operation by applying the data to the configuration datastore. During this processing, the following errors MUST be detected:

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

During <edit-config> processing:

- o If the NETCONF operation creates data nodes under a "choice", any existing nodes from other "case" branches are deleted by the server.
- o If the NETCONF operation modifies a data node such that any node's "when" expression becomes false, then the node with the "when" expression is deleted by the server.

#### 8.3.3. Validation

When datastore processing is complete, the final contents MUST obey all validation constraints. This validation processing is performed at differing times according to the datastore. If the datastore is <running/> or <startup/>, these constraints MUST be enforced at the end of the <edit-config> or <copy-config> operation. If the

datastore is <candidate/>, the constraint enforcement is delayed until a <commit> or <validate> operation.

- o Any "must" constraints MUST evaluate to "true".
- o Any referential integrity constraints defined via the "path" statement MUST be satisfied.
- o Any "unique" constraints on lists MUST be satisfied.
- o The "min-elements" and "max-elements" constraints are enforced for lists and leaf-lists.

## 9. Built-In Types

YANG has a set of built-in types, similar to those of many programming languages, but with some differences due to special requirements from the management information model.

Additional types may be defined, derived from those built-in types or from other derived types. Derived types may use subtyping to formally restrict the set of possible values.

The different built-in types and their derived types allow different kinds of subtyping, namely length and regular expression restrictions of strings (Section 9.4.4, Section 9.4.6) 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 12.

#### 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  $i \times 10^{-n}$  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  $i \times 10^{-n}$  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 9.4.6) 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 12.

#### 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. Usage Example

```
typedef my-base-str-type {
  type string {
    length "1..255";
  }
}

type my-base-str-type {
  // legal length refinement
  length "11 | 42..max"; // 11 | 42..255
}

type my-base-str-type {
  // illegal length refinement
  length "1..999";
}
```

#### 9.4.6. 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.

##### 9.4.6.1. 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
reference	7.19.4	0..1

#### 9.4.7. Usage Example

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
```

## 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, the leaf it refers to MUST also represent configuration. Such a leaf puts a constraint on valid data. All leafref nodes MUST reference existing leaf instances or leaves 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 cannot be restricted.

### 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 12.

The predicates are only used when more than one key reference is needed to uniquely identify a leaf instance. This occurs if a list has multiple keys, or a reference to a leaf other than the key in a list is needed. In these cases, multiple leafrefs are typically specified, and predicates are used to tie them together.

The "path" expression evaluates to a node set consisting of zero, one, or more nodes. If the leaf with the leafref type represents configuration data, this node set MUST be non-empty.

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

- o 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:

- o 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.
- o 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. Lexical Representation

A leafref value is encoded the same way as the leaf it references.

#### 9.9.4. Canonical Form

The canonical form of a leafref is the same as the canonical form of the leaf it references.

#### 9.9.5. 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.

#### 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";
    }
  }
}
```

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, except it MUST NOT be one of the built-in types "empty" or "leafref".

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.

Any default value or "units" property defined in the member types is not inherited by the union type.

Example:

```
type union {  
    type int32;  
    type enumeration {  
        enum "unbounded";  
    }  
}
```

##### 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.

### 9.13. 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 12. 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 9.13.2) 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:

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

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

- o 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.
- o 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 9.13.2).

#### 9.13.2. 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". 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.

#### 9.13.3. 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.

#### 9.13.4. Canonical Form

Since the lexical form depends on the XML context in which the value occurs, this type does not have a canonical form.

#### 9.13.5. 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. 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:

- o An "enumeration" type may have new enums added, provided the old enums's values do not change.
- o A "bits" type may have new bits added, provided the old bit positions do not change.
- o A "range", "length", or "pattern" statement may expand the allowed value space.
- o A "default" statement may be added to a leaf that does not have a default value (either directly or indirectly through its type).
- o A "units" statement may be added.
- o A "reference" statement may be added or updated.
- o A "must" statement may be removed or its constraint relaxed.
- o A "mandatory" statement may be removed or changed from "true" to "false".
- o A "min-elements" statement may be removed, or changed to require fewer elements.
- o A "max-elements" statement may be removed, or changed to allow more elements.
- o A "description" statement may be added or clarified without changing the semantics of the definition.
- o New typedefs, groupings, rpcs, notifications, extensions, features, and identities may be added.
- o New data definition statements may be added if they do not add mandatory nodes (Section 3.1) to existing nodes or at the top level in a module or submodule, or if they are conditionally dependent on a new feature (i.e., have an "if-feature" statement that refers to a new feature).
- o A new "case" statement may be added.
- o A node that represented state data may be changed to represent configuration, provided it is not mandatory (Section 3.1).
- o An "if-feature" statement may be removed, provided its node is not mandatory (Section 3.1).



- o A "status" statement may be added, or changed from "current" to "deprecated" or "obsolete", or from "deprecated" to "obsolete".
- o A "type" statement may be replaced with another "type" statement that does not change the syntax or semantics of the type. For example, an inline type definition may be replaced with a typedef, but an int8 type cannot be replaced by an int16, since the syntax would change.
- o Any set of data definition nodes may be replaced with another set of syntactically and semantically equivalent nodes. For example, a set of leafs may be replaced by a uses of a grouping with the same leafs.
- o A module may be split into a set of submodules, or a submodule may be removed, provided the definitions in the module do not change in any other way than allowed here.
- o The "prefix" statement may be changed, provided all local uses of the prefix also are changed.

Otherwise, if the semantics of any previous definition are changed (i.e., if a non-editorial change is made to any definition other than those specifically allowed above), then this **MUST** be achieved by a new definition with a new identifier.

In statements that have any data definition statements as substatements, those data definition substatements **MUST NOT** be reordered.

## 11. 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.

## 11.1.1. 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:

- o If the argument is represented as an attribute, this attribute has no namespace.
- o If the argument is represented as an element, it is qualified by the same namespace as its parent keyword element.
- o If the argument is represented as an element, it MUST be the first child of the keyword element.

Substatements of a YANG statement are represented as (additional) children of the keyword element and their relative order MUST be the same as the order of substatements in YANG.

Comments in YANG MAY be mapped to XML comments.

keyword	argument name	yin-element
anyxml	name	false
argument	name	false
augment	target-node	false
base	name	false

belongs-to	module	false
bit	name	false
case	name	false
choice	name	false
config	value	false
contact	text	true
container	name	false
default	value	false
description	text	true
deviate	value	false
deviation	target-node	false
enum	name	false
error-app-tag	value	false
error-message	value	true
extension	name	false
feature	name	false
fraction-digits	value	false
grouping	name	false
identity	name	false
if-feature	name	false
import	module	false
include	module	false
input	<no argument>	n/a
key	value	false
leaf	name	false
leaf-list	name	false
length	value	false
list	name	false
mandatory	value	false
max-elements	value	false
min-elements	value	false
module	name	false
must	condition	false
namespace	uri	false
notification	name	false
ordered-by	value	false
organization	text	true
output	<no argument>	n/a
path	value	false
pattern	value	false
position	value	false
prefix	value	false
presence	value	false
range	value	false
reference	text	true
refine	target-node	false
require-instance	value	false
revision	date	false

revision-date	date	false
rpc	name	false
status	value	false
submodule	name	false
type	name	false
typedef	name	false
unique	tag	false
units	name	false
uses	name	false
value	value	false
when	condition	false
yang-version	value	false
yin-element	value	false

Table 1: Mapping of arguments of the YANG statements.

## 11.1.1.1. 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: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>

```

## 12. 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 identifier-ref-arg-str
                 optsep stmtend

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
                          [error-message-stmt stmtsep]
                          [error-app-tag-stmt stmtsep]
                          [description-stmt stmtsep]
                          [reference-stmt stmtsep]
                        "}")
```

```
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 = path-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   = 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
                   *(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 abnngen'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'  
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'
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>

### 13. 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.

#### 13.1. 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").

### 13.2. 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.

### 13.3. 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.

### 13.4. 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

### 13.5. 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.
```

### 13.6. 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.
```

### 13.7. 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").

### 13.8. 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
```

#### 14. IANA Considerations

This document defines a registry for YANG module and submodule names. The name of the registry is "YANG Module Names".

The registry shall record for each entry:

- o the name of the module or submodule
- o for modules, the assigned XML namespace
- o for modules, the prefix of the module
- o for submodules, the name of the module it belongs to
- o a reference to the (sub)module's documentation (e.g., the RFC number)

There are no initial assignments.

For allocation, RFC publication is required as per RFC 5226 [RFC5226]. All registered YANG module names MUST comply with the rules for identifiers stated in Section 6.2, and MUST have a module name prefix.

The module name prefix 'ietf-' is reserved for IETF stream documents [RFC4844], while the module name prefix 'irtf-' is reserved for IRTF stream documents. Modules published in other RFC streams MUST have a similar suitable prefix.

All module and submodule names in the registry MUST be unique.

All XML namespaces in the registry MUST be unique.

This document registers two URIs for the YANG and YIN XML namespaces in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following have been registered.

URI: urn:ietf:params:xml:ns:yang:yin:1

URI: urn:ietf:params:xml:ns:yang:1

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces.

This document registers two new media types as defined in the following sections.

#### 14.1. 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>

#### 14.2. 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>



## 15. Security Considerations

This document defines a language with which to write and read descriptions of management information. The language itself has no security impact on the Internet.

The same considerations are relevant as for the base NETCONF protocol (see [RFC6241], Section 9).

Data modeled in YANG might contain sensitive information. RPCs or notifications defined in YANG might transfer sensitive information.

Security issues are related to the usage of data modeled in YANG. Such issues shall be dealt with in documents describing the data models and documents about the interfaces used to manipulate the data e.g., the NETCONF documents.

Data modeled in YANG is dependent upon:

- o the security of the transmission infrastructure used to send sensitive information.
- o the security of applications that store or release such sensitive information.
- o adequate authentication and access control mechanisms to restrict the usage of sensitive data.

YANG parsers need to be robust with respect to malformed documents. Reading malformed documents from unknown or untrusted sources could result in an attacker gaining privileges of the user running the YANG parser. In an extreme situation, the entire machine could be compromised.

## 16. 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)

## 17. 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.

## 18. ChangeLog

RFC Editor: remove this section upon publication as an RFC.

### 18.1. Version -00

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

## 19. References

### 19.1. 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>>.

## 19.2. 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](mailto:mbj@tail-f.com)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 18, 2014

A. Bierman  
YumaWorks  
June 16, 2014

Guidelines for Authors and Reviewers of YANG Data Model Documents  
draft-ietf-netmod-rfc6087bis-00

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 December 18, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
2. Terminology . . . . .	5
2.1. Requirements Notation . . . . .	5
2.2. NETCONF Terms . . . . .	5
2.3. YANG Terms . . . . .	5
2.4. Terms . . . . .	6
3. General Documentation Guidelines . . . . .	7
3.1. Module Copyright . . . . .	7
3.2. Terminology Section . . . . .	8
3.3. Tree Diagrams . . . . .	8
3.4. Narrative Sections . . . . .	9
3.5. Definitions Section . . . . .	9
3.6. Security Considerations Section . . . . .	9
3.7. IANA Considerations Section . . . . .	10
3.7.1. Documents that Create a New Namespace . . . . .	10
3.7.2. Documents that Extend an Existing Namespace . . . . .	10
3.8. Reference Sections . . . . .	10
4. YANG Usage Guidelines . . . . .	12
4.1. Module Naming Conventions . . . . .	12
4.2. Identifiers . . . . .	12
4.3. Defaults . . . . .	12
4.4. Conditional Statements . . . . .	13
4.5. XPath Usage . . . . .	13
4.6. Lifecycle Management . . . . .	15
4.7. Module Header, Meta, and Revision Statements . . . . .	15
4.8. Namespace Assignments . . . . .	16
4.9. Top-Level Data Definitions . . . . .	17
4.10. Data Types . . . . .	18
4.11. Reusable Type Definitions . . . . .	18
4.12. Data Definitions . . . . .	19
4.13. Operation Definitions . . . . .	20
4.14. Notification Definitions . . . . .	20
5. IANA Considerations . . . . .	21
6. Security Considerations . . . . .	22
6.1. Security Considerations Section Template . . . . .	22
7. Acknowledgments . . . . .	24
8. Changes Since RFC 6087 . . . . .	25
9. References . . . . .	26
9.1. Normative References . . . . .	26
9.2. Informative References . . . . .	26
Appendix A. Module Review Checklist . . . . .	28
Appendix B. YANG Module Template . . . . .	30

Author's Address . . . . .	32
----------------------------	----

## 1. Introduction

The standardization of network configuration interfaces for use with the Network Configuration Protocol [RFC6241] requires a modular set of data models, which can be reused and extended over time.

This document defines a set of usage guidelines for Standards Track documents containing [RFC6020] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF server. A server that supports a particular YANG module will support client NETCONF operation requests, as indicated by the specific content defined in the YANG module.

This document is similar to the Structure of Management Information version 2 (SMIV2) usage guidelines specification [RFC4181] in intent and structure. However, since that document was written a decade after SMIV2 modules had been in use, it was published as a 'Best Current Practice' (BCP). This document is not a BCP, but rather an informational reference, intended to promote consistency in documents containing YANG modules.

Many YANG constructs are defined as optional to use, such as the description statement. However, in order to maximize interoperability of NETCONF implementations utilizing YANG data models, it is desirable to define a set of usage guidelines that may require a higher level of compliance than the minimum level defined in the YANG specification.

In addition, YANG allows constructs such as infinite length identifiers and string values, or top-level mandatory nodes, that a compliant server is not required to support. Only constructs that all servers are required to support can be used in IETF YANG modules.

This document defines usage guidelines related to the NETCONF operations layer and NETCONF content layer, as defined in [RFC6241]. These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.



## 2. Terminology

### 2.1. Requirements Notation

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

RFC 2119 language is used here to express the views of the NETMOD working group regarding content for YANG modules. YANG modules complying with this document will treat the RFC 2119 terminology as if it were describing best current practices.

### 2.2. NETCONF Terms

The following terms are defined in [RFC6241] and are not redefined here:

- o capabilities
- o client
- o operation
- o server

### 2.3. YANG Terms

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

- o data node
- o module
- o namespace
- o submodule
- o version
- o YANG
- o YIN

Note that the term 'module' may be used as a generic term for a YANG module or submodule. When describing properties that are specific to submodules, the term 'submodule' is used instead.

## 2.4. Terms

The following terms are used throughout this document:

- o published: A stable release of a module or submodule, usually contained in an RFC.
- o unpublished: An unstable release of a module or submodule, usually contained in an Internet-Draft.

### 3. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors MUST be followed. The RFC Editor provides guidelines for authors of RFCs, which are first published as Internet-Drafts. These guidelines should be followed and are defined in [RFC2223] and updated in [RFC5741] and "RFC Document Style" [RFC-STYLE].

The following sections MUST be present in an Internet-Draft containing a module:

- o Narrative sections
- o Definitions section
- o Security Considerations section
- o IANA Considerations section
- o References section

#### 3.1. 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>

### 3.2. 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 as follows:

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

### 3.3. 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
```

### 3.4. 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.

### 3.5. 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 4 for guidelines on YANG usage.

### 3.6. 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 6.1 contains the security considerations template dated 2010-06-16. Authors **MUST** check the webpage at the URL listed above in case there is a more recent version available.

In particular:

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

- o Readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.
- o Operations (i.e., YANG 'rpc' statements) that are potentially harmful to system behavior or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

### 3.7. 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.

#### 3.7.1. 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.

#### 3.7.2. 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.

### 3.8. 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.

#### 4. 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.

##### 4.1. 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.

##### 4.2. 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].

##### 4.3. 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

#### 4.4. 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).

#### 4.5. XPath Usage

This section describes guidelines for using the XML Path Language [W3C.REC-xpath-19991116] (XPath) within YANG modules.

The 'attribute' and 'namespace' axes are not supported in YANG, and MAY be empty in a NETCONF server implementation.

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 '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 be 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'.

Data nodes that use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric 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 MUST NOT reference the context node or any descendant nodes of the context node.

#### 4.6. Lifecycle Management

The status statement **MUST** be present if its value is 'deprecated' or '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.

#### 4.7. 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 3.1.

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.

#### 4.8. Namespace Assignments

It is RECOMMENDED that only valid YANG modules be included in documents, whether or not they are published yet. This allows:

- o the module to compile correctly instead of generating disruptive fatal errors.
- o early implementors to use the modules without picking a random value for the XML namespace.
- o early interoperability testing since independent implementations will use the same XML namespace value.

Until a URI is assigned by the IANA, a proposed namespace URI MUST be provided for the namespace statement in a YANG module. A value SHOULD be selected that is not likely to collide with other YANG namespaces. Standard module names, prefixes, and URI strings already listed in the YANG Module Registry MUST NOT be used.

A standard namespace statement value SHOULD have the following form:

```
<URN prefix string>:<module-name>
```

The following URN prefix string SHOULD be used for published and unpublished YANG modules:

```
urn:ietf:params:xml:ns:yang:
```

The following example URNs would be valid temporary namespace

statement values for Standards Track modules:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-partial-lock
```

```
urn:ietf:params:xml:ns:yang:ietf-netconf-state
```

```
urn:ietf:params:xml:ns:yang:ietf-netconf
```

Note that a different URN prefix string SHOULD be used for non-Standards-Track modules. The string SHOULD be selected according to the guidelines in [RFC6020].

The following examples of non-Standards-Track modules are only suggestions. There are no guidelines for this type of URN in this document:

```
http://example.com/ns/example-interfaces
```

```
http://example.com/ns/example-system
```

#### 4.9. 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.

#### 4.10. 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.

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.

#### 4.11. 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.

#### 4.12. Data Definitions

The description statement MUST be present in the following YANG statements:

- o anyxml
- o augment
- o choice
- o container
- o extension
- o feature
- o grouping
- o identity
- o leaf
- o leaf-list
- o list
- o notification
- o rpc
- o typedef

If the data definition semantics are defined in an external document, (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

The 'anyxml' construct may be useful to represent an HTML banner containing markup elements, such as '<b>' and '</b>', and MAY be used in such cases. However, this construct SHOULD NOT be

used if other YANG data node types can be used instead to represent the desired syntax and semantics.

If there are referential integrity constraints associated with the desired semantics that can be represented with XPath, then one or more 'must' statements SHOULD be present.

For list and leaf-list data definitions, if the number of possible instances is required to be bounded for all implementations, then the max-elements statements SHOULD be present.

If any 'must' or 'when' statements are used within the data definition, then the data definition description statement SHOULD describe the purpose of each one.

#### 4.13. 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.

#### 4.14. 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.



## 5. 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 B.

Field	Value
Name	ietf-template
Namespace	urn:ietf:params:xml:ns:yang:ietf-template
Prefix	temp
Reference	RFC XXXX

### YANG Registry Assignment

## 6. 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.

### 6.1. 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>

## 7. 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, and Ladislav Lhotka for their extensive reviews and contributions to this document.

## 8. Changes Since RFC 6087

The following changes have been made to the guidelines published in [RFC6087]:

- o Updated NETCONF reference from RFC 4741 to RFC 6241
- o Updated NETCONF over SSH citation from RFC 4742 to RFC 6242
- o Updated YANG Types reference from RFC 6021 to RFC 6991
- o Updated obsolete URLs for IETF resources
- o Changed top-level data node guideline
- o Clarified XPath usage for a literal value representing a YANG identity
- o Clarified XPath usage for a when-stmt
- o Added terminology guidelines
- o Added YANG tree diagram guideline

## 9. References

### 9.1. Normative References

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

### 9.2. 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. Module Review Checklist

This section is adapted from RFC 4181.

The purpose of a YANG module review is to review the YANG module both for technical correctness and for adherence to IETF documentation requirements. The following checklist may be helpful when reviewing an Internet-Draft:

- o I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see <http://www.ietf.org/id-info/guidelines.html>), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.
- o Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in <http://www.ietf.org/id-info/guidelines.html>.
- o Copyright Notice -- verify that the draft has the appropriate text regarding the rights that document contributors provide to the IETF Trust [RFC5378]. Verify that it contains the full IETF Trust copyright notice at the beginning of the document. The IETF Trust Legal Provisions (TLP) can be found at:

<http://trustee.ietf.org/license-info/>

- o Security Considerations section -- verify that the draft uses the latest approved template from the OPS area website (<http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines>) and that the guidelines therein have been followed.
- o IANA Considerations section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

XML Namespace Registry: Register the YANG module namespace.

YANG Module Registry: Register the YANG module name, prefix, namespace, and RFC number, according to the rules specified in [RFC6020].



- o References -- verify that the references are properly divided between normative and informative references, that RFC 2119 is included as a normative reference if the terminology defined therein is used in the document, that all references required by the boilerplate are present, that all YANG modules containing imported items are cited as normative references, and that all citations point to the most current RFCs unless there is a valid reason to do otherwise (for example, it is OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module).
- o License -- verify that the draft contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in Section 3.1. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

<http://trustee.ietf.org/license-info/>

- o Other Issues -- check for any issues mentioned in <http://www.ietf.org/id-info/checklist.html> that are not covered elsewhere.
- o Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

<http://trac.tools.ietf.org/wg/netconf/trac/wiki>

Checking for correct syntax, however, is only part of the job. It is just as important to actually read the YANG module document from the point of view of a potential implementor. It is particularly important to check that description statements are sufficiently clear and unambiguous to allow interoperable implementations to be created.

## Appendix B. 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](mailto:andy@yumaworks.com)



NETMOD  
Internet-Draft  
Intended status: Standards Track  
Expires: November 26, 2014

L. Lhotka  
CZ.NIC  
May 25, 2014

A YANG Data Model for Routing Management  
draft-ietf-netmod-routing-cfg-15

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 individual routing protocols and other related 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 November 26, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
2. Terminology and Notation . . . . .	5
2.1. Glossary of New Terms . . . . .	5
2.2. Tree Diagrams . . . . .	6
2.3. Prefixes in Data Node Names . . . . .	6
3. Objectives . . . . .	8
4. The Design of the Core Routing Data Model . . . . .	9
4.1. System-Controlled and User-Controlled List Entries . . . . .	12
4.2. Features of Advanced Routers . . . . .	13
5. Basic Building Blocks . . . . .	14
5.1. Routing Instance . . . . .	14
5.1.1. Parameters of IPv6 Routing Instance Interfaces . . . . .	15
5.2. Route . . . . .	16
5.3. Routing Information Base (RIB) . . . . .	16
5.3.1. Multiple RIBs per Address Family . . . . .	17
5.4. Routing Protocol . . . . .	17
5.4.1. Routing Pseudo-Protocols . . . . .	18
5.4.2. Defining New Routing Protocols . . . . .	20
5.5. Route Filter . . . . .	21
5.6. RPC Operations . . . . .	22
6. Interactions with Other YANG Modules . . . . .	23
6.1. Module "ietf-interfaces" . . . . .	23
6.2. Module "ietf-ip" . . . . .	23
7. Routing Management YANG Module . . . . .	25
8. IPv4 Unicast Routing Management YANG Module . . . . .	47
9. IPv6 Unicast Routing Management YANG Module . . . . .	54
10. IANA Considerations . . . . .	69
11. Security Considerations . . . . .	71
12. Acknowledgments . . . . .	72
13. References . . . . .	73
13.1. Normative References . . . . .	73
13.2. Informative References . . . . .	73
Appendix A. The Complete Data Trees . . . . .	74
A.1. Configuration Data . . . . .	74
A.2. Operational State Data . . . . .	76
Appendix B. Minimum Implementation . . . . .	78
Appendix C. Example: Adding a New Routing Protocol . . . . .	79
Appendix D. Example: NETCONF <get> Reply . . . . .	82
Appendix E. Change Log . . . . .	89
E.1. Changes Between Versions -14 and -15 . . . . .	89
E.2. Changes Between Versions -13 and -14 . . . . .	89

E.3.	Changes Between Versions -12 and -13 . . . . .	89
E.4.	Changes Between Versions -11 and -12 . . . . .	90
E.5.	Changes Between Versions -10 and -11 . . . . .	90
E.6.	Changes Between Versions -09 and -10 . . . . .	90
E.7.	Changes Between Versions -08 and -09 . . . . .	91
E.8.	Changes Between Versions -07 and -08 . . . . .	91
E.9.	Changes Between Versions -06 and -07 . . . . .	91
E.10.	Changes Between Versions -05 and -06 . . . . .	91
E.11.	Changes Between Versions -04 and -05 . . . . .	92
E.12.	Changes Between Versions -03 and -04 . . . . .	93
E.13.	Changes Between Versions -02 and -03 . . . . .	93
E.14.	Changes Between Versions -01 and -02 . . . . .	94
E.15.	Changes Between Versions -00 and -01 . . . . .	94
Author's Address	. . . . .	95



## 1. Introduction

This document contains a specification of the following YANG modules:

- o Module "ietf-routing" provides generic components of a routing data model.
- o Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.
- o Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast, including the router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is proposed as a basis for the development of data models for configuration and management of 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 setups involving multiple routing protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

## 2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- o client
- o message
- o protocol operation
- o server

The following terms are defined in [RFC6020]:

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

### 2.1. Glossary of New Terms

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 operational 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 operational state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

## 2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node and "\*" denotes a "list" or "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2.3. Prefixes in Data Node Names

In this document, names of data nodes, RPC 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	[YANG-IP]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

### 3. Objectives

The initial design of the core routing data model was driven by the following objectives:

- o The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).
- o Simple routing setups, such as static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.
- o On the other hand, the core routing framework must allow for complicated setups involving multiple routing information bases (RIB) and multiple routing protocols, as well as controlled redistributions of routing information.
- o Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

#### 4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Figures 1 and 2 show abridged views of the configuration and operational 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* [name]
  |   |   |   +--rw name
  |   |   |   +--rw description?
  |   |   |   +--rw enabled?
  |   |   |   +--rw type
  |   |   |   +--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 router-id?
  |   +--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* [name]
  |   |   |   +--ro name
  |   |   |   +--ro type
  |   |   |   +--ro connected-ribs
  |   |   |   ...
  +--ro ribs
  |   +--ro rib* [name]
  |   |   +--ro name
  |   |   +--ro id
  |   |   +--ro address-family
  |   |   +--ro routes
  |   |   |   +--ro route* [id]
  |   |   |   ...
  |   +--ro recipient-ribs
  |   |   +--ro recipient-rib* [rib-name]
  |   |   ...
  +--ro route-filters
  |   +--ro route-filter* [name]
  |   |   +--ro name
  |   |   +--ro type

```

Figure 2: Operational 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.



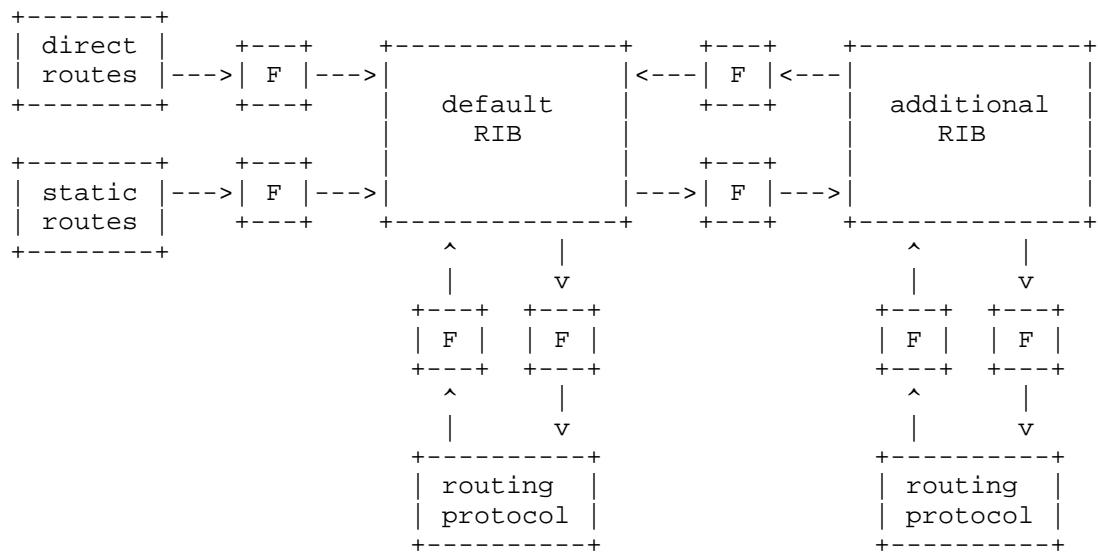


Figure 3: Example setup of a routing system

The example in Figure 3 shows a typical (though certainly not the only possible) organization of a more complex routing subsystem for a single address family. Several of its features are worth mentioning:

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

Additional entries may be created in the configuration by the user 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 operational state version of the list.

Corresponding entries in both versions of the list (in operational 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 operational state 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 operational state 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 operational state 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 advanced functionality optional by means of two YANG features:

- o "multiple-ribs" - indicates that the device supports multiple RIBs per address family, routing protocols connected to non-default RIBs, and RIBs configured as receivers of routes from other RIBs.
- o "multipath-routes" - indicates that the device supports 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

Each routing instance in the core routing data model represents a logical router. The exact semantics of this term are left to implementations. For example, routing instances may be completely isolated virtual routers or, alternatively, they may internally share certain information.

A routing instance together with its operational state is represented as an entry of the list `"/routing-state/routing-instance"`, and identified by a unique name. Configuration of that router instance appears as an entry of the list `"/routing/routing-instance"`.

An implementation MAY support multiple types of logical routers simultaneously. Instances of all routing instance types are organized as entries of the same flat `"routing-instance"` list. In order to discriminate routing instances belonging to different types, the `"type"` leaf is defined as a child of the `"routing-instance"` node.

An implementation MAY create one or more system-controlled routing instances, and MAY also pose restrictions on allowed routing instance types and on the 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:standard-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].

In YANG terms, the list of routing instance interfaces is modeled as a `"list"` node rather than `"leaf-list"` in order to allow for adding, via augmentation, other configuration or state data related to the corresponding interface.

Implementations MAY specify additional rules for the assignment of interfaces to routing instances. For example, it may be required

that the sets of interfaces assigned to different routing instances be disjoint.

#### 5.1.1.1. 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 operational state data, with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

- o send-advertisements,
- o max-rtr-adv-interval,
- o min-rtr-adv-interval,
- o managed-flag,
- o other-config-flag,
- o link-mtu,
- o reachable-time,
- o retrans-timer,
- o cur-hop-limit,
- o default-lifetime,
- o prefix-list: a list of prefixes to be advertised.

The following parameters are associated with each prefix in the list:

- \* valid-lifetime,
- \* on-link-flag,
- \* preferred-lifetime,
- \* autonomous-flag.

The definitions and descriptions of the above parameters can be found in the module "ietf-ipv6-unicast-routing" (Section 9).

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [YANG-IP] (leaf "ip:forwarding").
2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-unicast-routing" module therefore assumes the former behavior with constant values.

## 5.2. Route

Routes are basic elements of information in a routing system. The core routing data model defines only the following minimal set of route attributes:

- o destination prefix: IP prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.
- o next-hop or action: outgoing interface, IP address of one or more adjacent routers to which a packet should be forwarded, or a special action such as discarding the packet.

The above list of route attributes suffices for a simple static routing configuration. It is expected that future modules defining routing protocols will add other route attributes such as metrics or preferences.

Routes and their attributes are used both in configuration data, for example as manually configured static routes, and in operational state data, for example as entries in RIBs.

## 5.3. Routing Information Base (RIB)

A routing information base (RIB) is a list of routes complemented with administrative data, namely:

- o "source-protocol": type of the routing protocol from which the route was originally obtained.
- o "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 operational 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, an implementation MAY specify rules and restrictions for sharing RIBs among routing instances.

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).

Each routing protocol instance is connected to exactly one RIB for each address family that the routing protocol instance supports. Routes learned from the network by a routing protocol are normally installed into the connected RIB(s) and, conversely, routes from the connected RIB(s) are normally injected into the routing protocol. However, routing protocol implementations MAY specify rules that restrict this exchange of routes in either direction (or both directions).

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:

- o import filter controls which routes are passed from the routing protocol instance to the connected RIB,
- o 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 that they are confined to the local device and do not exchange any routing information with neighboring routers. Routes from both "direct" and "static" protocol instances are passed to the connected RIB (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* [id]
|   |   +--rw v4ur:id
|   |   +--rw v4ur:description?
|   |   +--rw v4ur:destination-prefix
|   |   +--rw (next-hop-options)
|   |   |   +--:(special-next-hop)
|   |   |   |   +--rw v4ur:special-next-hop?
|   |   |   +--:(simple-next-hop)
|   |   |   |   +--rw v4ur:next-hop?
|   |   |   |   +--rw v4ur:outgoing-interface?
|   |   +--:(next-hop-list) {rt:multipath-routes}?
|   |   +--rw v4ur:next-hop-list
|   |   |   +--rw v4ur:next-hop* [id]
|   |   |   |   +--rw v4ur:id
|   |   |   |   +--rw v4ur:address?
|   |   |   |   +--rw v4ur:outgoing-interface?
|   |   |   |   +--rw v4ur:priority?
|   |   |   |   +--rw v4ur:weight?
+--rw v6ur:ipv6
+--rw v6ur:route* [id]
+--rw v6ur:id
+--rw v6ur:description?
+--rw v6ur:destination-prefix
+--rw (next-hop-options)
+--:(special-next-hop)
|   +--rw v6ur:special-next-hop?
+--:(simple-next-hop)
|   +--rw v6ur:next-hop?
|   +--rw v6ur:outgoing-interface?
+--:(next-hop-list) {rt:multipath-routes}?
+--rw v6ur:next-hop-list
+--rw v6ur:next-hop* [id]
+--rw v6ur:id
+--rw v6ur:address?
+--rw v6ur:outgoing-interface?
+--rw v6ur:priority?
+--rw v6ur:weight?

```

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

#### 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:active-route/rt:output/rt:route,

and possibly other places in the configuration, state data and RPC input or output.

- o Configuration parameters and/or state data for the new protocol can be defined by augmenting the "routing-protocol" data node under both "/routing" and "/routing-state".
- o Per-interface configuration, including activation of the routing protocol on individual interfaces, can use references to entries in the list of routing instance interfaces (rt:interface).

By using the "when" statement, the augmented configuration parameters and state data specific to the new protocol **SHOULD** be made conditional and valid only if the value of "rt:type" or "rt:source-protocol" is equal to the new protocol's identity. It is also **RECOMMENDED** that protocol-specific data nodes be encapsulated in appropriately named containers.

The above steps are implemented by the example YANG module for the RIP routing protocol in Appendix C.

## 5.5. 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, an implementation **MAY** specify rules and restrictions for sharing route filters among routing instances.

By itself, the route filtering framework defined in this document allows for applying only two extreme routing policies which are represented by the following pre-defined route filter types:

- o "deny-all-route-filter": all routes are blocked,
- o "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 is identified by a unique name. Its type MUST be specified by the "type" identity reference - this opens the space for multiple route filtering framework implementations.

#### 5.6. RPC Operations

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

- o active-route: query a routing instance for the active route that is currently used for sending datagrams to a destination host whose address is passed as an input parameter.
- o route-count: retrieve the total number of entries in a RIB.

## 6. Interactions with Other YANG Modules

The semantics of the core routing data model also depend 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 [YANG-IP]:

```
/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 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 "ietf-routing@2014-05-24.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 2014-05-24 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Features */

feature multiple-ribs {
  description
    "This feature indicates that the device supports multiple RIBS
    per address family, and the framework for passing routes
    between RIBs.

    Devices that do not support this feature MUST provide exactly
    one system-controlled RIB per supported address family. These
    RIBs then appear as entries of the list
    /routing-state/ribs/rib.";
}

feature multipath-routes {
  description
    "This feature indicates that the device supports multipath
    routes that have a list of next-hops.";
}

/* 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-type {
  description
    "Base identity from which identities describing routing
    instance types are derived.

    It is primarily intended for discriminating among different
    types of logical routers or router virtualization.";
}

identity standard-routing-instance {
  base routing-instance-type;
  description
    "This identity represents a default routing instance.";
}

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 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 a route filter in
      state data.";
  }

  /* 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 defines 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 the definition of router ID.";  
    leaf router-id {  
      type yang:dotted-quad;  
      description  
        "Router ID - 32-bit number in the form of a dotted quad. Some  
        protocols use this parameter for identifying a router to its  
        neighbors.";  
    }  
  }  
  
  grouping outgoing-interface {  
    description  
      "This grouping defines the outgoing interface for use in  
      next-hops.";  
    leaf outgoing-interface {  
      type leafref {  
        path "/rt:routing-state/rt:routing-instance/rt:interfaces/"  
          + "rt:interface/rt:name";  
      }  
      description  
        "Name of the outgoing interface.";  
    }  
  }  
  
  grouping special-next-hop {  
    description  
      "This grouping provides the leaf for specifying special  
      next-hop options.";  
    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 network
            device.";
    }
}
description
    "Special next-hop options.";
}
}

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 next-hop-content {  
    description  
      "Generic parameters of next-hops in routes.";  
    choice next-hop-options {  
      mandatory "true";  
      description  
        "Options for expressing the next-hop in routes.";  
      case special-next-hop {  
        uses special-next-hop;  
      }  
      case simple-next-hop {  
        uses outgoing-interface;  
      }  
      case next-hop-list {  
        if-feature multipath-routes;  
        container next-hop-list {  
          description  
            "Container for multiple next-hops.";  
          list next-hop {  
            key "id";  
            description  
              "An entry of a next-hop list.";  
            uses state-entry-id;  
            uses outgoing-interface;  
            uses next-hop-classifiers;  
          }  
        }  
      }  
    }  
  }  
  
  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 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";
        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-type;
            }
            description

```

```
    "The routing instance type, primarily intended for
    discriminating among different types of logical routers,
    route virtualization, master-slave arrangements etc.,
    while keeping all routing instances in the same flat
    list.";
  }
  uses router-id {
    description
      "Global router ID.

      An implementation may choose a value if none is
      configured.

      Routing protocols that use router ID MAY override this
      global parameter.";
  }
  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 "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 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 type {
            type identityref {
                base routing-protocol;
            }
            mandatory "true";
            description
                "Type of the routing protocol.";
        }
    }
}
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 (at most one RIB per address
            family).";
        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 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 contents of the RIB.";
        list route {
            key "id";
            description
                "A RIB route entry. This data node MUST be augmented
                with information specific for routes of each address
                family.";
            uses state-entry-id;
            uses next-hop-content;
            uses route-metadata;
        }
    }
    container recipient-ribs {
        if-feature multiple-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-type;
  }
  default "rt:standard-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 {
  description
    "Configuration of the global router ID.";
}
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 "name";
        description
            "Each entry contains configuration of a routing protocol
            instance.";
        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 type {
        type identityref {
            base routing-protocol;
        }
        mandatory "true";
        description
            "Type of the routing protocol - an identity derived
            from the 'routing-protocol' base identity.";
    }
    container connected-ribs {
        description
            "Configuration of connected RIBs.";
        list connected-rib {
            must "not(/routing/ribs/rib[name=current()/"
                + "preceding-sibling::connected-rib/"
                + "rib-name and address-family=/routing/ribs/"
                + "rib[name=current()/rib-name]/address-family])" {
                error-message
                    "Duplicate address family for connected RIBs.";
                description
                    "For each address family, there MUST NOT be more
                    than one connected RIB.";
            }
            key "rib-name";
            description
                "List of RIBs to which the routing protocol instance
                is connected (at most one RIB per address family).

                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
    "Configured 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 active-route {
    description
        "Return the active 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;
    uses next-hop-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 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 "ietf-ipv4-unicast-routing@2014-05-24.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 2014-05-24 {
  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-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 {
      type inet:ipv4-address;
      description
        "IPv4 address of the next-hop.";
    }
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
    + "rt:next-hop-options/rt:next-hop-list/rt:next-hop-list/"
    + "rt:next-hop" {
    when "../.../rt:address-family = 'v4ur:ipv4-unicast'" {
      description
        "This augment is valid only for IPv4 unicast.";
    }
    if-feature rt:multipath-routes;
    description
      "This leaf augments the 'next-hop-list' case of IPv4 unicast
      routes.";
    leaf 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 "id";
      ordered-by "user";
      description
        "A user-ordered list of static routes.";
      leaf id {
        type uint32 {
          range "1..max";
        }
        description
          "Unique numeric identifier of the route."
      }
    }
  }
}
```

```
        This value is unrelated to system-assigned 'id'
        parameters of routes in RIBs.";
    }
    leaf description {
        type string;
        description
            "Textual description of the route.";
    }
    leaf destination-prefix {
        type inet:ipv4-prefix;
        mandatory "true";
        description
            "IPv4 destination prefix.";
    }
    choice next-hop-options {
        mandatory "true";
        description
            "Options for expressing the next-hop in static routes.";
        case special-next-hop {
            uses rt:special-next-hop;
        }
        case simple-next-hop {
            leaf next-hop {
                type inet:ipv4-address;
                description
                    "IPv4 address of the next-hop.";
            }
            leaf outgoing-interface {
                type leafref {
                    path "../.../.../rt:interfaces/rt:interface/"
                        + "rt:name";
                }
                description
                    "Name of the outgoing interface.

                    Only interfaces configured for the ancestor routing
                    instance can be given.";
            }
        }
    }
    case next-hop-list {
        if-feature rt:multipath-routes;
        container next-hop-list {
            description
                "Configuration of multiple next-hops.";
            list next-hop {
                key "id";
                description
                    "An entry of a next-hop list.";
            }
        }
    }
}
```

```

    leaf id {
      type uint32;
      description
        "Unique numeric identifier of the entry.

        This value is unrelated to system-assigned 'id'
        parameters of next-hops in RIBs.";
    }
    leaf address {
      type inet:ipv4-address;
      description
        "IPv4 address of the next-hop.";
    }
    leaf outgoing-interface {
      type leafref {
        path "../../../../../rt:interfaces/"
          + "rt:interface/rt:name";
      }
      description
        "Name of the outgoing interface.

        Only interfaces configured for the ancestor
        routing instance can be given.";
    }
    uses rt:next-hop-classifiers {
      refine "priority" {
        default "primary";
      }
      refine "weight" {
        default "0";
      }
    }
  }
}

/* RPC methods */

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.";
  }
  description

```



```
    "This leaf augments the 'rt:destination-address' parameter of
      the 'rt:active-route' operation.";
  leaf address {
    type inet:ipv4-address;
    description
      "IPv4 destination address.";
  }
}

augment "/rt:active-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:active-route'
      operation.";
  leaf destination-prefix {
    type inet:ipv4-prefix;
    description
      "IPv4 destination prefix.";
  }
}

augment "/rt:active-route/rt:output/rt:route/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:active-route' operation.";
  leaf next-hop {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

augment "/rt:active-route/rt:output/rt:route/rt:next-hop-options/"
  + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
  when "../rt:address-family='v4ur:ipv4-unicast'" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  if-feature rt:multipath-routes;
  description
```

```
        "This leaf augments the 'next-hop-list' case in the reply to
          the 'rt:active-route' operation.";
    leaf address {
        type inet:ipv4-address;
        description
            "IPv4 address of the next-hop.";
    }
}
}
}

<CODE ENDS>
```

## 9. IPv6 Unicast Routing Management 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 "ietf-ipv6-unicast-routing@2014-05-25.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 2014-05-25 {
  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-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:ribs/rt:rib/rt:routes/rt:route/"
  + "rt:next-hop-options/rt:next-hop-list/rt:next-hop-list/"
  + "rt:next-hop" {
  when "../.../rt:address-family = 'v6ur:ipv6-unicast'" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  if-feature rt:multipath-routes;
  description
    "This leaf augments the 'next-hop-list' case of IPv6 unicast
    routes.";
  leaf 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 "id";
      ordered-by "user";
      description
        "A user-ordered list of static routes.";
      leaf id {
        type uint32 {
          range "1..max";
        }
        description
          "Unique numeric identifier of the route.

          This value is unrelated to system-assigned 'id'
          parameters of routes in RIBs.";
      }
      leaf description {
        type string;
        description
          "Textual description of the route.";
      }
      leaf destination-prefix {
        type inet:ipv6-prefix;
        mandatory "true";
        description
          "IPv6 destination prefix.";
      }
      choice next-hop-options {
        mandatory "true";
        description
          "Options for expressing the next-hop in static routes.";
        case special-next-hop {
          uses rt:special-next-hop;
        }
        case simple-next-hop {
          leaf next-hop {
            type inet:ipv6-address;
            description
              "IPv6 address of the next-hop.";
          }
          leaf outgoing-interface {
            type leafref {
              path "../.../.../.../rt:interfaces/rt:interface/"
                + "rt:name";
            }
          }
        }
      }
    }
  }
}
```

```
    }
    description
      "Name of the outgoing interface.

      Only interfaces configured for the ancestor routing
      instance can be given.";
  }
}
case next-hop-list {
  if-feature rt:multipath-routes;
  container next-hop-list {
    description
      "Configuration of multiple next-hops.";
    list next-hop {
      key "id";
      description
        "An entry of a next-hop list.";
      leaf id {
        type uint32;
        description
          "Unique numeric identifier of the entry.

          This value is unrelated to system-assigned 'id'
          parameters of next-hops in RIBs.";
      }
      leaf address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
      leaf outgoing-interface {
        type leafref {
          path "../../../../../rt:interfaces/"
            + "rt:interface/rt:name";
        }
        description
          "Name of the outgoing interface.

          Only interfaces configured for the ancestor
          routing instance can be given.";
      }
    }
  }
  uses rt:next-hop-classifiers {
    refine "priority" {
      default "primary";
    }
    refine "weight" {
      default "0";
    }
  }
}
```

```

    }
  }
}

/* RPC methods */

augment "/rt:active-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:active-route' operation.";
  leaf address {
    type inet:ipv6-address;
    description
      "IPv6 destination address.";
  }
}

augment "/rt:active-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:active-route'
    operation.";
  leaf destination-prefix {
    type inet:ipv6-prefix;
    description
      "IPv6 destination prefix.";
  }
}

augment "/rt:active-route/rt:output/rt:route/rt:next-hop-options/"
  + "rt:simple-next-hop" {
  when "rt:address-family='v6ur:ipv6-unicast'" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description

```



```
        "This leaf augments the 'simple-next-hop' case in the reply to
          the 'rt:active-route' operation.";
    leaf next-hop {
        type inet:ipv6-address;
        description
            "IPv6 address of the next-hop.";
    }
}

augment "/rt:active-route/rt:output/rt:route/rt:next-hop-options/"
    + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
    when "../rt:address-family='v6ur:ipv6-unicast'" {
        description
            "This augment is valid only for IPv6 unicast.";
    }
    if-feature rt:multipath-routes;
    description
        "This leaf augments the 'next-hop-list' case in the reply to
          the 'rt:active-route' operation.";
    leaf 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, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Thomas Morin, Tom Petch, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler and Yi Yang 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., Ed., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, September 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "NETCONF Configuration Protocol", RFC 6241, June 2011.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.
- [YANG-IP] Bjorklund, M., "A YANG Data Model for IP Management", draft-ietf-netmod-ip-cfg-14 (work in progress), March 2014.

### 13.2. Informative References

- [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 operational 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* [name]
  +--rw name                string
  +--rw description?        string
  +--rw enabled?            boolean
  +--rw type                 identityref
  +--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* [id]
        +--rw v4ur:id                uint32
        +--rw v4ur:description?     string
        +--rw v4ur:destination-prefix inet:ipv4-prefix
        +--rw (next-hop-options)
          +--:(special-next-hop)
          | +--rw v4ur:special-next-hop? enumeration
          +--:(simple-next-hop)
          | +--rw v4ur:next-hop?      inet:ipv4-address
          | +--rw v4ur:outgoing-interface? leafref
          +--:(next-hop-list) {rt:multipath-routes}?
            +--rw v4ur:next-hop-list
              +--rw v4ur:next-hop* [id]
                +--rw v4ur:id                uint32
                +--rw v4ur:address? inet:ipv4-address
                +--rw v4ur:outgoing-interface? leafref
                +--rw v4ur:priority?      enumeration
                +--rw v4ur:weight?        uint8
    +--rw v6ur:ipv6
      +--rw v6ur:route* [id]
        +--rw v6ur:id                uint32
        +--rw v6ur:description?     string
        +--rw v6ur:destination-prefix inet:ipv6-prefix
        +--rw (next-hop-options)
          +--:(special-next-hop)
          | +--rw v6ur:special-next-hop? enumeration
          +--:(simple-next-hop)
          | +--rw v6ur:next-hop?      inet:ipv6-address
          | +--rw v6ur:outgoing-interface? leafref
          +--:(next-hop-list) {rt:multipath-routes}?
            +--rw v6ur:next-hop-list
              +--rw v6ur:next-hop* [id]
                +--rw v6ur:id                uint32
                +--rw v6ur:address? inet:ipv6-address
                +--rw v6ur:outgoing-interface? leafref
                +--rw v6ur:priority?      enumeration

```



```

|                                     |--rw v6ur:weight?          uint8
+--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. Operational State Data

```

+--ro routing-state
|   +--ro routing-instance* [name]
|   |   +--ro name                string
|   |   +--ro id                  uint64
|   |   +--ro type?               identityref
|   |   +--ro router-id?          yang:dotted-quad
|   |   +--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* [name]
|   |   +--ro name          string
|   |   +--ro type          identityref
|   |   +--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 ribs
|   +--ro rib* [name]
|   |   +--ro name          string
|   |   +--ro id            uint64
|   |   +--ro address-family identityref
|   |   +--ro routes
|   |   |   +--ro route* [id]
|   |   |   |   +--ro id            uint64
|   |   |   |   +--ro (next-hop-options)
|   |   |   |   |   +--:(special-next-hop)
|   |   |   |   |   |   +--ro special-next-hop?    enumeration
|   |   |   |   |   +--:(simple-next-hop)
|   |   |   |   |   |   +--ro outgoing-interface?  leafref
|   |   |   |   |   |   +--ro v4ur:next-hop?       inet:ipv4-address
|   |   |   |   |   |   +--ro v6ur:next-hop?       inet:ipv6-address
|   |   |   |   |   +--:(next-hop-list) {multipath-routes}?
|   |   |   |   |   +--ro next-hop-list
|   |   |   |   |   |   +--ro next-hop* [id]
|   |   |   |   |   |   |   +--ro id            uint64
|   |   |   |   |   |   |   +--ro outgoing-interface? leafref
|   |   |   |   |   |   |   +--ro priority?       enumeration
|   |   |   |   |   |   |   +--ro weight?        uint8
|   |   |   |   |   |   |   +--ro v4ur:address?   inet:ipv4-address
|   |   |   |   |   |   |   +--ro v6ur:address?   inet:ipv6-address
|   |   |   |   +--ro source-protocol    identityref
|   |   |   +--ro last-updated?          yang:date-and-time
|   |   |   +--ro v4ur:destination-prefix? inet:ipv4-prefix
|   |   |   +--ro v6ur:destination-prefix? inet:ipv6-prefix
+--ro recipient-ribs {multiple-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:

- o 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 they will also appear as system-controlled entries of the "default-rib" list in operational state data. No user-controlled RIBs are allowed.
- o 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, we do not follow 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 &lt;get&gt; 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:

- o ietf-interfaces [RFC7223],
- o ietf-ip [YANG-IP],
- o ietf-routing (Section 7),
- o ietf-ipv4-unicast-routing (Section 8),
- o ietf-ipv6-unicast-routing (Section 9).

We assume a simple network setup 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.

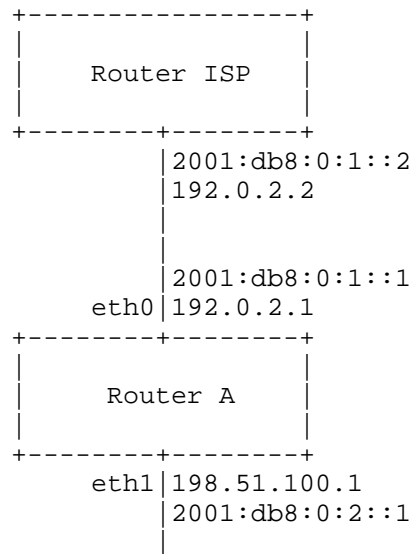


Figure 5: Example network configuration

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>
        2013-07-02T17:11:27+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>
        2013-07-02T17:11:27+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:name>st0</rt:name>
        <rt:description>
          Static routing is used for the internal network.
        </rt:description>
        <rt:type>rt:static</rt:type>
        <rt:static-routes>
          <v4ur:ipv4>
            <v4ur:route>
              <v4ur:id>1</v4ur:id>
              <v4ur:destination-prefix>0.0.0.0/0</v4ur:destination-prefix>
              <v4ur:next-hop>192.0.2.2</v4ur:next-hop>
            </v4ur:route>
          </v4ur:ipv4>
        </rt:static-routes>
      </rt:routing-protocol>
    </rt:routing-protocols>
  </rt:routing-instance>
</rt:routing>
```

```
</v4ur:ipv4>
<v6ur:ipv6>
  <v6ur:route>
    <v6ur:id>1</v6ur:id>
    <v6ur:destination-prefix>::/0</v6ur:destination-prefix>
    <v6ur:next-hop>2001:db8:0:1::2</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:router-id>192.0.2.1</rt:router-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:name>st0</rt:name>
        <rt:type>rt:static</rt:type>
```

```
</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>
        <rt:id>626433832</rt:id>
        <v4ur:destination-prefix>
          192.0.2.1/24</v4ur:destination-prefix>
        <rt:outgoing-interface>eth0</rt:outgoing-interface>
        <rt:source-protocol>rt:direct</rt:source-protocol>
        <rt:last-updated>2013-07-02T17:11:27+01:00</rt:last-updated>
      </rt:route>
      <rt:route>
        <rt:id>795028841</rt:id>
        <v4ur:destination-prefix>
          198.51.100.0/24</v4ur:destination-prefix>
        <rt:outgoing-interface>eth1</rt:outgoing-interface>
        <rt:source-protocol>rt:direct</rt:source-protocol>
        <rt:last-updated>2013-07-02T17:11:27+01:00</rt:last-updated>
      </rt:route>
      <rt:route>
        <rt:id>971693993</rt:id>
        <v4ur:destination-prefix>0.0.0.0/0</v4ur:destination-prefix>
        <rt:source-protocol>rt:static</rt:source-protocol>
        <v4ur:next-hop>192.0.2.2</v4ur:next-hop>
        <rt:last-updated>2013-07-02T18:02:45+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>
        <rt:id>749445923</rt:id>
        <v6ur:destination-prefix>
          2001:db8:0:1::/64</v6ur:destination-prefix>
        <rt:outgoing-interface>eth0</rt:outgoing-interface>
        <rt:source-protocol>rt:direct</rt:source-protocol>
        <rt:last-updated>2013-07-02T17:11:27+01:00</rt:last-updated>
      </rt:route>
      <rt:route>

```

```
<rt:id>78164062</rt:id>
<v6ur:destination-prefix>
  2001:db8:0:2::/64</v6ur:destination-prefix>
<rt:outgoing-interface>eth1</rt:outgoing-interface>
<rt:source-protocol>rt:direct</rt:source-protocol>
<rt:last-updated>2013-07-02T17:11:27+01:00</rt:last-updated>
</rt:route>
<rt:route>
  <rt:id>862089986</rt:id>
  <v6ur:destination-prefix>::/0</v6ur:destination-prefix>
  <v6ur:next-hop>2001:db8:0:1::2</v6ur:next-hop>
  <rt:source-protocol>rt:static</rt:source-protocol>
  <rt:last-updated>2013-07-02T18:02:45+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 -14 and -15

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

## E.2. Changes Between Versions -13 and -14

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

## E.3. Changes Between Versions -12 and -13

- o Wrote appendix about minimum implementation.
- o Remove "when" statement for IPv6 router interface operational state - it was dependent on a config value that may not be present.
- o Extra container for the next-hop list.
- o Names rather than numeric ids are used for referring to list entries in operational state.
- o Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- o Descriptions of "name" keys in operational state lists are required to be persistent.
- o
- o Removed "if-feature multiple-ribs;" from connected-ribs.
- o "rib-name" instead of "name" is used as the name of leafref nodes.

- o "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

#### E.4. Changes Between Versions -11 and -12

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

#### E.5. Changes Between Versions -10 and -11

- o Migrated address families from IANA enumerations to identities.
- o Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- o Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- o Described the relationship between system-controlled and user-controlled list entries.
- o Feature "user-defined-routing-tables" changed into "advanced-router".
- o Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).
- o Added nexthop-list with entries having priorities (backup) and weights (load balancing).
- o Updated bibliography references.

#### E.6. Changes Between Versions -09 and -10

- o Added subtree for operational state data ("/routing-state").
- o Terms "system-controlled entry" and "user-controlled entry" defined and used.

- o New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.
- o Added grouping "router-id".
- o In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".
- o Renamed "main-routing-table" to "default-routing-table".

#### E.7. Changes Between Versions -08 and -09

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

#### E.8. Changes Between Versions -07 and -08

- o Changed reference from RFC6021 to RFC6021bis.

#### E.9. Changes Between Versions -06 and -07

- o The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.
- o The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.
- o The default behavior for IPv6 RA prefix advertisements was clarified.
- o Changed type of "rt:router-id" to "ip:dotted-quad".
- o Type of "rt:router-id" changed to "yang:dotted-quad".
- o Fixed missing prefixes in XPath expressions.

#### E.10. Changes Between Versions -05 and -06

- o Document title changed: "Configuration" was replaced by "Management".



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

#### E.11. Changes Between Versions -04 and -05

- o Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- o "must" statement for "static-routes" changed to "when".
- o Added "main-routing-tables" containing references to main routing tables for each address family.
- o Removed the defaults for "address-family" and "safi" and made them mandatory.
- o Removed the default for route-filter/type and made this leaf mandatory.
- o If there is no active route for a given destination, the "active-route" RPC returns no output.
- o Added "enabled" switch under "routing-protocol".
- o Added "router-type" identity and "type" leaf under "router".
- o Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- o The "direct" pseudo-protocol is always connected to main routing tables.
- o Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".

- o Added "must" constraint saying that a routing table must not be its own recipient.

#### E.12. Changes Between Versions -03 and -04

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

#### E.13. Changes Between Versions -02 and -03

- o Module "iana-afn-safi" moved to I-D "iana-if-type".
- o Removed forwarding table.
- o RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- o New RPC "route-count".
- o For both RPCs, specification of negative responses was added.
- o Relaxed separation of router instances.
- o Assignment of interfaces to router instances needn't be disjoint.
- o Route filters are now global.
- o Added "allow-all-route-filter" for symmetry.
- o Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- o Added "router-id" leaf.
- o Specified the names for IPv4/IPv6 unicast main routing tables.
- o Route parameter "last-modified" changed to "age".

- o Added container "recipient-routing-tables".

#### E.14. Changes Between Versions -01 and -02

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

#### E.15. Changes Between Versions -00 and -01

- o AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- o Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.
- o Names of some data nodes were changed, in particular "routing-process" is now "router".
- o The restriction of a single AFN/SAFI per router was lifted.
- o RPC operation "delete-route" was removed.
- o Illegal XPath references from "get-route" to the datastore were fixed.
- o Section "Security Considerations" was written.

Author's Address

Ladislav Lhotka  
CZ.NIC

Email: [lhotka@nic.cz](mailto:lhotka@nic.cz)



NETMOD  
Internet-Draft  
Intended status: Standards Track  
Expires: October 23, 2014

L. Lhotka  
CZ.NIC  
April 21, 2014

JSON Encoding of Data Modeled with YANG  
draft-ietf-netmod-yang-json-00

Abstract

This document defines rules for representing configuration and state data defined using YANG as JSON text. It does so by specifying a procedure for translating the subset of YANG-compatible XML documents to JSON text, and vice versa. A JSON encoding of XML attributes is also defined so as to allow for including metadata in JSON documents.

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 October 23, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology and Notation . . . . .	4
3. Specification of the Translation Procedure . . . . .	5
3.1. Names and Namespaces . . . . .	6
3.2. Mapping XML Elements to JSON Objects . . . . .	8
3.2.1. The "leaf" Data Node . . . . .	8
3.2.2. The "container" Data Node . . . . .	8
3.2.3. The "leaf-list" Data Node . . . . .	9
3.2.4. The "list" Data Node . . . . .	9
3.2.5. The "anyxml" Data Node . . . . .	10
3.3. Mapping YANG Datatypes to JSON Values . . . . .	11
3.3.1. Numeric Datatypes . . . . .	11
3.3.2. The "string" Type . . . . .	11
3.3.3. The "boolean" Type . . . . .	11
3.3.4. The "enumeration" Type . . . . .	11
3.3.5. The "bits" Type . . . . .	12
3.3.6. The "binary" Type . . . . .	12
3.3.7. The "leafref" Type . . . . .	12
3.3.8. The "identityref" Type . . . . .	12
3.3.9. The "empty" Type . . . . .	12
3.3.10. The "union" Type . . . . .	13
3.3.11. The "instance-identifier" Type . . . . .	13
4. Encoding Metadata in JSON . . . . .	14
5. IANA Considerations . . . . .	16
6. Security Considerations . . . . .	16
7. Acknowledgments . . . . .	17
8. References . . . . .	17
8.1. Normative References . . . . .	17
8.2. Informative References . . . . .	17
Appendix A. A Complete Example . . . . .	18
Author's Address . . . . .	20

## 1. Introduction

The aim of this document is define rules for representing configuration and state data defined using the YANG data modeling language [RFC6020] as JavaScript Object Notation (JSON) text [RFC7159]. The result can be potentially applied in two different ways:

1. JSON may be used instead of the standard XML [XML] encoding in the context of the NETCONF protocol [RFC6241] and/or with existing data models expressed in YANG. An example application is the RESTCONF Protocol [RESTCONF].

2. Other documents that choose JSON to represent structured data can use YANG for defining the data model, i.e., both syntactic and semantic constraints that the data have to satisfy.

JSON mapping rules could be specified in a similar way as the XML mapping rules in [RFC6020]. This would however require solving several problems. To begin with, YANG uses XPath [XPath] quite extensively, but XPath is not defined for JSON and such a definition would be far from straightforward.

In order to avoid these technical difficulties, this document employs an alternative approach: it defines a relatively simple procedure which allows for translating the subset of XML that can be modeled using YANG to JSON, and vice versa. Consequently, validation of a JSON text against a data model can be done by translating the JSON text to XML, which is then validated according to the rules stated in [RFC6020].

The translation procedure is adapted to YANG specifics and requirements, namely:

1. The translation is driven by a concrete YANG data model and uses information about data types to achieve better results than generic XML-JSON translation procedures.
2. Various document types are supported, namely configuration data, configuration + state data, RPC input and output parameters, and notifications.
3. XML namespaces specified in the data model are mapped to namespaces of JSON objects. However, explicit namespace identifiers are rarely needed in JSON text.
4. Section 4 defines JSON encoding of XML attributes. Although XML attributes cannot be modeled with YANG, they are often used for attaching metadata to elements, and a standard JSON encoding is therefore needed.
5. Translation of XML mixed content, comments and processing instructions is outside the scope of this document.

Item 1 above also means that, depending on the data model, the same XML element can be translated to different JSON objects. For example,

```
<foo>123</foo>
```

is translated to



```
"foo": 123
```

if the "foo" node is defined as a leaf with the "uint8" datatype, or to

```
"foo": ["123"]
```

if the "foo" node is defined as a leaf-list with the "string" datatype, and the <foo> element has no siblings of the same name.

## 2. Terminology and Notation

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

The following terms are defined in [RFC6020]:

- o anyxml
- o augment
- o container
- o data node
- o data tree
- o datatype
- o feature
- o identity
- o instance identifier
- o leaf
- o leaf-list
- o list
- o module
- o submodule

The following terms are defined in [XMLNS]:

- o local name
- o prefixed name
- o qualified name

### 3. Specification of the Translation Procedure

The translation procedure defines a 1-1 correspondence between the subset of YANG-compatible XML documents and JSON text. This means that the translation can be applied in both directions and it is always invertible.

The translation procedure is applicable only to data hierarchies that are modelled by a YANG data model. An input XML document MAY contain enclosing elements representing NETCONF "Operations" and "Messages" layers. However, these enclosing elements do not appear in the resulting JSON document.

Any YANG-compatible XML document can be translated, except documents with mixed content. This is only a minor limitation since mixed content is marginal in YANG - it is allowed only in anyxml data nodes.

The following sections specify rules mainly for translating XML documents to JSON text. Rules for the inverse translation are stated only where necessary, otherwise they can be easily inferred.

REQUIRED parameters of the translation procedure are:

- o YANG data model consisting of a set of YANG modules,
- o type of the input document,
- o optional features (defined via the "feature" statement) that are considered active.

The permissible types of input documents are listed in Table 1 together with the corresponding part of the data model that is used for the translation.

Document Type	Data Model Section
configuration and state data	main data tree
configuration	main data tree ("config true")
RPC input parameters	"input" data nodes under "rpc"
RPC output parameters	"output" data nodes under "rpc"
notification	"notification" data nodes

Table 1: YANG Document Types

When translating XML to JSON, the type of the input document can often be determined from the encapsulating elements belonging to the "Operations" or "Messages" layer as defined by the NETCONF protocol (see Sec. 1.2 in [RFC6241]).

A particular application MAY decide to support only a subset of document types from Table 1.

XML documents can be translated to JSON text only if they are valid instances of the YANG data model and selected document type, also taking into account the active features, if there are any.

The resulting JSON document is always a single object ([RFC7159], Sec. 4) whose members are translated from the original XML document using the rules specified in the following sections.

### 3.1. Names and Namespaces

The local part of a JSON name is always identical to the local name of the corresponding XML element.

Each JSON name lives in a namespace which is uniquely identified by the name of the YANG module where the corresponding data node is defined. If the data node is defined in a submodule, then the namespace identifier is the name of the main module to which the submodule belongs. The translation procedure MUST correctly map YANG namespace URIs to YANG module names and vice versa.

The namespace SHALL be expressed in JSON text by prefixing the local name in the following way:

<module name>:<local name>

Figure 1: Encoding a namespace identifier with a local name.

The namespace identifier **MUST** be used for local names that are ambiguous, i.e., whenever the data model permits a sibling data node with the same local name. Otherwise, the namespace identifier is **OPTIONAL**.

For example, consider the following YANG module:

```
module foomod {
  namespace "http://example.com/foomod";
  prefix "fm";
  container foo {
    leaf bar {
      type boolean;
    }
  }
}
```

If the data model consists only of this module, then the following is a valid JSON document:

```
{
  "foo": {
    "bar": true
  }
}
```

Now, assume the container "foo" is augmented from another module:

```
module barmod {
  namespace "http://example.com/barmod";
  prefix "bm";
  import foomod {
    prefix fm;
  }
  augment "/fm:foo" {
    leaf bar {
      type uint8;
    }
  }
}
```

In the data model combining "foomod" and "barmod", we have two sibling data nodes with the same local name, namely "bar". In this

case, a valid JSON document has to specify an explicit namespace identifier (module name) for both leaves:

```
{
  "foo": {
    "foomod:bar": true,
    "barmod:bar": 123
  }
}
```

### 3.2. Mapping XML Elements to JSON Objects

An XML element that is modelled as a YANG data node is translated to a name/value pair where the name is formed from the name of the XML element using the rules in Section 3.1. The value depends on the type of the data node as specified in the following sections.

#### 3.2.1. The "leaf" Data Node

An XML element that is modeled as YANG leaf is translated to a name/value pair and the type of the value is derived from the YANG datatype of the leaf (see Section 3.3 for the datatype mapping rules).

Example: For the leaf node definition

```
leaf foo {
    type uint8;
}
```

the XML element

```
<foo>123</foo>
```

corresponds to the JSON name/value pair

```
"foo": 123
```

#### 3.2.2. The "container" Data Node

An XML element that is modeled as YANG container is translated to a name/object pair.

Example: For the container definition

```
    container bar {  
      leaf foo {  
        type uint8;  
      }  
    }
```

the XML element

```
<bar>  
  <foo>123</foo>  
</bar>
```

corresponds to the JSON name/value pair

```
"bar": {  
  "foo": 123  
}
```

### 3.2.3. The "leaf-list" Data Node

A sequence of one or more sibling XML elements with the same qualified name that is modeled as YANG leaf-list is translated to a name/array pair, and the array elements are primitive values whose type depends on the datatype of the leaf-list (see Section 3.3).

Example: For the leaf-list definition

```
    leaf-list foo {  
      type uint8;  
    }
```

the XML elements

```
<foo>123</foo>  
<foo>0</foo>
```

correspond to the JSON name/value pair

```
"foo": [123, 0]
```

### 3.2.4. The "list" Data Node

A sequence of one or more sibling XML elements with the same qualified name that is modeled as YANG list is translated to a name/array pair, and the array elements are JSON objects.

Unlike the XML encoding, where the list keys are required to come before any other siblings, and in the order specified by the data

model, the order of members within a JSON list entry is arbitrary, because JSON objects are fundamentally unordered collections of members.

Example: For the list definition

```
list bar {  
  key foo;  
  leaf foo {  
    type uint8;  
  }  
  leaf baz {  
    type string;  
  }  
}
```

the XML elements

```
<bar>  
  <foo>123</foo>  
  <baz>zig</baz>  
</bar>  
<bar>  
  <foo>0</foo>  
  <baz>zag</baz>  
</bar>
```

correspond to the JSON name/value pair

```
"bar": [  
  {  
    "foo": 123,  
    "baz": "zig"  
  },  
  {  
    "foo": 0,  
    "baz": "zag"  
  }  
]
```

### 3.2.5. The "anyxml" Data Node

An XML element that is modeled as a YANG anyxml data node is translated to a name/object pair. The content of such an element is not modelled by YANG, and there may not be a straightforward mapping to JSON text (e.g., if it is a mixed XML content). Therefore, translation of anyxml contents is necessarily application-specific and outside the scope of this document.

Example: For the anyxml definition

```
anyxml bar;
```

the XML element

```
<bar>
  <p xmlns="http://www.w3.org/1999/xhtml">
    This is <em>very</em> cool.
  </p>
</bar>
```

may be translated to the following JSON name/value pair:

```
{
  "bar": {
    "p": "This is very cool."
  }
}
```

### 3.3. Mapping YANG Datatypes to JSON Values

#### 3.3.1. Numeric Datatypes

A value of one of the YANG numeric datatypes ("int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64" and "decimal64") is mapped to a JSON number using the same lexical representation.

#### 3.3.2. The "string" Type

A "string" value is mapped to an identical JSON string, subject to JSON encoding rules.

#### 3.3.3. The "boolean" Type

A "boolean" value is mapped to the corresponding JSON value 'true' or 'false'.

#### 3.3.4. The "enumeration" Type

An "enumeration" value is mapped in the same way as a string except that the permitted values are defined by "enum" statements in YANG.



### 3.3.5. The "bits" Type

A "bits" value is mapped to a string identical to the lexical representation of this value in XML, i.e., space-separated names representing the individual bit values that are set.

### 3.3.6. The "binary" Type

A "binary" value is mapped to a JSON string identical to the lexical representation of this value in XML, i.e., base64-encoded binary data.

### 3.3.7. The "leafref" Type

A "leafref" value is mapped according to the same rules as the type of the leaf being referred to.

### 3.3.8. The "identityref" Type

An "identityref" value is mapped to a string representing the qualified name of the identity. Its namespace MAY be expressed as shown in Figure 1. If the namespace part is not present, the namespace of the name of the JSON object containing the value is assumed.

### 3.3.9. The "empty" Type

An "empty" value is mapped to '[null]', i.e., an array with the 'null' value being its only element.

This encoding was chosen instead of using simply 'null' in order to facilitate the use of empty leafs in common programming languages. When used in a boolean context, the '[null]' value, unlike 'null', evaluates to 'true'.

Example: For the leaf definition

```
leaf foo {  
    type empty;  
}
```

the XML element

```
<foo/>
```

corresponds to the JSON name/value pair

```
"foo": [null]
```

### 3.3.10. The "union" Type

YANG "union" type represents a choice among multiple alternative types. The actual type of the XML value MUST be determined using the procedure specified in Sec. 9.12 of [RFC6020] and the mapping rules for that type are used.

For example, consider the following YANG definition:

```
leaf-list bar {  
  type union {  
    type uint16;  
    type string;  
  }  
}
```

The sequence of three XML elements

```
<bar>6378</bar>  
<bar>14.5</bar>  
<bar>infinity</bar>
```

will then be translated to this name/array pair:

```
"bar": [6378, "14.5", "infinity"]
```

### 3.3.11. The "instance-identifier" Type

An "instance-identifier" value is a string representing a simplified XPath specification. It is mapped to an analogical JSON string in which all occurrences of XML namespace prefixes are either removed or replaced with the corresponding module name according to the rules of Section 3.1.

When translating such a value from JSON to XML, all components of the instance-identifier MUST be given appropriate XML namespace prefixes. It is RECOMMENDED that these prefixes be those defined via the "prefix" statement in the corresponding YANG modules.

For example, assume "ex" is the prefix defined for the "example" module. Then the XML-encoded instance identifier

```
/ex:system/ex:user[ex:name='fred']
```

corresponds to the following JSON-encoded instance identifier:

```
/example:system/example:user[example:name='fred']
```

or simply

```
/system/user[name='fred']
```

if the local names of the data nodes "system", "user" and "name" are unambiguous.

#### 4. Encoding Metadata in JSON

By design, YANG does not allow for modeling XML attributes. However, attributes are often used in XML instance documents for attaching various types of metadata information to elements. It is therefore desirable to have a standard way for representing attributes in JSON documents as well.

The metadata encoding defined in the rest of this section satisfies the following two important requirements:

1. There has to be a way for adding metadata to instances of all types of YANG data nodes, i.e., leafs, containers, list and leaf-list entries, and anyxml nodes.
2. The encoding of YANG data node instances as defined in the previous sections must not change.

Existing proposals for metadata encoding in JSON, such as [JSON-META], are oriented on rather specific uses of metadata, and fall short with respect to the first requirement.

All attributes assigned to an XML element are mapped in JSON to members (name/value pairs) of a single object, henceforth denoted as the metadata object. The placement of this object depends on the type of the element from YANG viewpoint, as specified in the following paragraphs.

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

Examples:

- o If "cask" is a container or anyxml node, the XML instance with attributes

```
<cask foo="a" bar="b">
  ...
</cask>
```

is mapped to the following JSON object:

```
"cask": {  
  "@": {  
    "foo": "a",  
    "bar": "b"  
  }  
  ...  
}
```

- o If "seq" is a list, then the pair of XML elements

```
<seq foo="a">  
  <name>one</name>  
</seq>  
<seq bar="b">  
  <name>two</name>  
</seq>
```

is mapped to the following JSON array:

```
"seq": [  
  {  
    "@": {  
      "foo": "a"  
    },  
    "name": "one"  
  },  
  {  
    "@": {  
      "bar": "b"  
    },  
    "name": "two"  
  }  
]
```

In order to assign attributes to a leaf instance, a sibling name/value pair is added, where the name is the symbol "@" concatenated with the identifier of the leaf.

For example, the element

```
<flag foo="a" bar="b">true</flag>
```

is mapped to the following two name/value pairs:

```
"flag": true,  
"@flag": {  
  "foo": "a",  
  "bar": "b"  
}
```

Finally, for a leaf-list instance, which is represented as a JSON array with primitive values, attributes may be assigned to one or more entries by adding a sibling name/value pair, 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 attributes assigned to the i-th entry of the leaf-list, or nil if the i-th entry has no attributes.

Trailing nil values in the array, i.e., those following the last non-nil metadata object, MAY be omitted.

For example, a leaf-list instance with four entries

```
<folio>6</folio>  
<folio foo="a">3</folio>  
<folio bar="b">7</folio>  
<folio>8</folio>
```

is mapped to the following two name/value pairs:

```
"folio": [6, 3, 7, 8],  
"@folio": [nil, {"foo": "a"}, {"bar": "b"}]
```

The encoding of attributes as specified above has the following two limitations:

- o Mapping of namespaces of XML attributes is undefined.
- o Attribute values can only be strings, other data types are not supported.

## 5. IANA Considerations

TBD - register application/yang.data+json media type?

## 6. Security Considerations

TBD.

## 7. Acknowledgments

The author wishes to thank Andy Bierman, Martin Bjorklund and Phil Shafer for their helpful comments and suggestions.

## 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.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, September 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "NETCONF Configuration Protocol", RFC 6241, June 2011.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.
- [XMLNS] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "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>>.
- [XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2006/REC-xml-20060816>>.

### 8.2. Informative References

- [IF-CFG] Bjorklund, M., "A YANG Data Model for Interface Management", draft-ietf-netmod-interfaces-cfg-16 (work in progress), January 2014.
- [JSON-META] Sakimura, N., "JSON Metadata", draft-sakimura-json-metadata-01 (work in progress), November 2013.

## [RESTCONF]

Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-ietf-netconf-restconf-00 (work in progress), March 2014.

## [XPath]

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

## Appendix A. A Complete Example

The JSON document shown below was translated from a reply to the NETCONF <get> request that can be found in Appendix D of [IF-CFG]. The data model is a combination of two YANG modules: "ietf-interfaces" and "ex-vlan" (the latter is an example module from Appendix C of [IF-CFG]). The "if-mib" feature defined in the "ietf-interfaces" module is considered to be active.

```
{
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "vlan-tagging": true
      },
      {
        "name": "eth1.10",
        "type": "iana-if-type:l2vlan",
        "enabled": true,
        "base-interface": "eth1",
        "vlan-id": 10
      },
      {
        "name": "lo1",
        "type": "iana-if-type:softwareLoopback",
        "enabled": true
      }
    ]
  },
  "interfaces-state": {
```

```
"interface": [  
  {  
    "name": "eth0",  
    "type": "iana-if-type:ethernetCsmacd",  
    "admin-status": "down",  
    "oper-status": "down",  
    "if-index": 2,  
    "phys-address": "00:01:02:03:04:05",  
    "statistics": {  
      "discontinuity-time": "2013-04-01T03:00:00+00:00"  
    }  
  },  
  {  
    "name": "eth1",  
    "type": "iana-if-type:ethernetCsmacd",  
    "admin-status": "up",  
    "oper-status": "up",  
    "if-index": 7,  
    "phys-address": "00:01:02:03:04:06",  
    "higher-layer-if": [  
      "eth1.10"  
    ],  
    "statistics": {  
      "discontinuity-time": "2013-04-01T03:00:00+00:00"  
    }  
  },  
  {  
    "name": "eth1.10",  
    "type": "iana-if-type:l2vlan",  
    "admin-status": "up",  
    "oper-status": "up",  
    "if-index": 9,  
    "lower-layer-if": [  
      "eth1"  
    ],  
    "statistics": {  
      "discontinuity-time": "2013-04-01T03:00:00+00:00"  
    }  
  },  
  {  
    "name": "eth2",  
    "type": "iana-if-type:ethernetCsmacd",  
    "admin-status": "down",  
    "oper-status": "down",  
    "if-index": 8,  
    "phys-address": "00:01:02:03:04:07",  
    "statistics": {  
      "discontinuity-time": "2013-04-01T03:00:00+00:00"  
    }  
  }  
]
```



```
    }  
  },  
  {  
    "name": "lo1",  
    "type": "iana-if-type:softwareLoopback",  
    "admin-status": "up",  
    "oper-status": "up",  
    "if-index": 1,  
    "statistics": {  
      "discontinuity-time": "2013-04-01T03:00:00+00:00"  
    }  
  }  
]  
}  
}
```

## Author's Address

Ladislav Lhotka  
CZ.NIC

Email: lhotka@nic.cz

ISIS Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 29, 2014

S. Litkowski  
Orange  
June 27, 2014

Yang Data Model for ISIS protocol  
draft-litkowski-isis-yang-isis-cfg-01

## Abstract

This document defines a YANG data model that can be used to configure and manage ISIS protocol.

## Requirements Language

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on December 29, 2014.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Tree diagram . . . . .	2
2. Design of the data model . . . . .	3
2.1. ISIS configuration . . . . .	5
2.2. Multitopology parameters . . . . .	5
2.3. Per level parameters . . . . .	5
2.4. Per interface parameters . . . . .	6
2.5. Operational states . . . . .	7
3. RPC operations . . . . .	8
4. Notifications . . . . .	8
5. Interaction with other YANG modules . . . . .	9
6. Yang module . . . . .	9
7. Security Considerations . . . . .	45
8. Acknowledgements . . . . .	46
9. IANA Considerations . . . . .	46
10. Normative References . . . . .	46
Appendix A. Example: NETCONF <get> Reply . . . . .	47
Author's Address . . . . .	53

## 1. Introduction

This document defines a YANG data model for ISIS routing protocol.

The data model covers configuration of an ISIS routing protocol instance as well as operational states.

### 1.1. Tree diagram

A simplified graphical representation of the data model is presented in Section 2. The meaning of the symbols in these diagrams is as follows:

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

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2. Design of the data model

The ISIS Yang module is divided in two main containers :

- o isis-cfg : that contains writable configuration objects.
- o isis-state : that contains read-only states.

The container isis-cfg and isis-state are augmenting the "routing-protocol" lists in ietf-routing module with specific ISIS parameters.

The figure below describe the overall structure of the isis Yang module :

```

module: isis
augment /rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route:
  +--ro metric?          uint32
  +--ro tag*             uint32
  +--ro route-type?     enumeration

augment /rt:active-route/rt:output/rt:route:
  +--ro metric?          uint32
  +--ro tag*             uint32
  +--ro route-type?     enumeration

augment
  /rt:routing/rt:routing-instance/rt:routing-protocols/rt:routing-protocol:
    +--rw isis
      +--rw isis-level?          isis-level
      +--rw nsap-address         simple-iso-address
      +--rw ipv4-router-id?      inet:ipv4-address
      +--rw ipv6-router-id?      inet:ipv6-address
      +--rw reference-bandwidth? uint32
      +--rw lsp-mtu?            uint16
      +--rw lsp-lifetime?       uint16
      +--rw lsp-refresh?        uint16
      +--rw psnp-authentication? boolean
      +--rw csnp-authentication? boolean
      +--rw hello-authentication? boolean
      +--rw authentication-key?  string
      +--rw authentication-type? enumeration
      +--rw isis-multi-topology-cfg
      |   ...
      +--rw isis-level-1-cfg
      |   ...

```

```

    +--rw isis-level-2-cfg
    |   ...
    +--rw overload
    |   +--rw status?      boolean
    |   +--rw timeout?     uint16
    +--rw interfaces
    |   +--rw interface* [name]
    |   ...

augment
/rt:routing-state/rt:routing-instance/rt:routing-protocols/rt:routing-protocol:
  +--ro isis-state
  |   +--ro adjacencies
  |   |   +--ro adjacency* [interface]
  |   |   |   +--ro interface      string
  |   |   |   +--ro level?        isis-level
  |   |   |   +--ro state?        enumeration
  |   +--ro spf-log
  |   |   +--ro event* [id]
  |   |   |   +--ro id              uint32
  |   |   |   +--ro spf-type?      enumeration
  |   |   |   +--ro level?        isis-level
  |   |   |   +--ro spf-delay?    uint32
  |   |   |   +--ro schedule-timestamp? yang:timestamp
  |   |   |   +--ro start-timestamp? yang:timestamp
  |   |   |   +--ro end-timestamp?  yang:timestamp
  |   |   |   +--ro trigger-lsp* [lsp]
  |   |   |   |   +--ro lsp        isis-lsp-id
  |   |   |   |   +--ro sequence?  uint32
  |   +--ro lsp-log
  |   |   +--ro event* [id]
  |   |   |   +--ro id              uint32
  |   |   |   +--ro level?        isis-level
  |   |   |   +--ro lsp
  |   |   |   |   +--ro lsp?      isis-lsp-id
  |   |   |   |   +--ro sequence? uint32
  |   |   +--ro received-timestamp? yang:timestamp
  +--ro database
  |   +--ro level-1
  |   |   ...
  |   +--ro level-2
  |   |   ...
  +--ro hostnames
  |   +--ro hostname* [system-id]
  |   |   +--ro system-id    isis-system-id
  |   |   +--ro hostname?   string
```

## 2.1. ISIS configuration

The ISIS configuration container is divided in :

- o Global parameters.
- o Level specific parameters (see Section 2.3).
- o Per interface configuration (see Section 2.4).

## 2.2. Multitopology parameters

The multitopology section is used to enable support of MT extensions for specific address families.

A boolean is associated with each topology type, defining if the topology is enabled or not.

```
+-rw isis-multi-topology-cfg
|   +-rw ipv4-unicast?      boolean
|   +-rw ipv6-unicast?      boolean
|   +-rw ipv4-multicast?    boolean
|   +-rw ipv6-multicast?    boolean
```

## 2.3. Per level parameters

The level parameter section of the ISIS instance describes the global parameters for level 1 and 2 including authentication, protocol preferences, default metrics ...

Level specific parameters override parameters contained directly under isis-cfg container.

```

+--rw isis-level-1-cfg
|   +--rw enabled?                boolean
|   +--rw psnp-authentication?    boolean
|   +--rw csnp-authentication?    boolean
|   +--rw hello-authentication?   boolean
|   +--rw authentication-key?     string
|   +--rw authentication-type?    enumeration
|   +--rw metric-type?            enumeration
|   +--rw preference?             uint8
|   +--rw external-preference?    uint8
|   +--rw default-ipv4-unicast-metric? isis-wide-metric
|   +--rw default-ipv6-unicast-metric? isis-wide-metric
|   +--rw default-ipv4-multicast-metric? isis-wide-metric
|   +--rw default-ipv6-multicast-metric? isis-wide-metric
+--rw isis-level-2-cfg
|   +--rw enabled?                boolean
|   +--rw psnp-authentication?    boolean
|   +--rw csnp-authentication?    boolean
|   +--rw hello-authentication?   boolean
|   +--rw authentication-key?     string
|   +--rw authentication-type?    enumeration
|   +--rw metric-type?            enumeration
|   +--rw preference?             uint8
|   +--rw external-preference?    uint8
|   +--rw default-ipv4-unicast-metric? isis-wide-metric
|   +--rw default-ipv6-unicast-metric? isis-wide-metric
|   +--rw default-ipv4-multicast-metric? isis-wide-metric
|   +--rw default-ipv6-multicast-metric? isis-wide-metric

```

#### 2.4. Per interface parameters

The per-interface section of the ISIS instance describes the interface specific parameters.

Each interface has interface-specific parameters and level-specific parameters. A level-specific parameter always override interface-specific parameter and an interface-specific parameter always override an isis global parameter (defined in isis-cfg).

```

+--rw interfaces
|   +--rw interface* [name]
|       +--rw name                leafref
|       +--rw level?              isis-level
|       +--rw lsp-interval?       uint16
|       +--rw passive?            boolean
|       +--rw csnp-interval?      uint16
|       +--rw hello-authentication-type? enumeration
|       +--rw hello-authentication-key? string

```

```

+--rw hello-interval?          uint16
+--rw hello-multiplier?        uint16
+--rw hello-padding?           boolean
+--rw ipv4-unicast?            boolean
+--rw ipv6-unicast?            boolean
+--rw ipv4-multicast?          boolean
+--rw ipv6-multicast?          boolean
+--rw interface-type?          enumeration
+--rw enabled?                 boolean
+--rw tag*                      uint32
+--rw level-1
|   +--rw hello-authentication-type?  enumeration
|   +--rw hello-authentication-key?   string
|   +--rw hello-interval?             uint16
|   +--rw hello-multiplier?           uint16
|   +--rw ipv4-unicast?               boolean
|   +--rw ipv6-unicast?               boolean
|   +--rw ipv4-multicast?             boolean
|   +--rw ipv6-multicast?             boolean
|   +--rw priority?                  uint8
|   +--rw ipv4-unicast-metric?        isis-wide-metric
|   +--rw ipv6-unicast-metric?        isis-wide-metric
|   +--rw ipv4-multicast-metric?      isis-wide-metric
|   +--rw ipv6-multicast-metric?      isis-wide-metric
|   +--rw passive?                   boolean
+--rw level-2
|   +--rw hello-authentication-type?  enumeration
|   +--rw hello-authentication-key?   string
|   +--rw hello-interval?             uint16
|   +--rw hello-multiplier?           uint16
|   +--rw ipv4-unicast?               boolean
|   +--rw ipv6-unicast?               boolean
|   +--rw ipv4-multicast?             boolean
|   +--rw ipv6-multicast?             boolean
|   +--rw priority?                  uint8
|   +--rw ipv4-unicast-metric?        isis-wide-metric
|   +--rw ipv6-unicast-metric?        isis-wide-metric
|   +--rw ipv4-multicast-metric?      isis-wide-metric
|   +--rw ipv6-multicast-metric?      isis-wide-metric
|   +--rw passive?                   boolean

```

## 2.5. Operational states

isis-state container provides operational states for ISIS. This container is divided in multiple components :

- o adjacencies : provides state information about current ISIS adjacencies.



- o spf-log : provides information about SPF events on the node.
- o lsp-log : provides information about LSP events on the node (reception of an LSP or modification of local LSP).
- o database : provides details on current LSDB.
- o hostname : provides information about system-id to hostname mappings.

### 3. RPC operations

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

- o clear-isis-database : reset the content of a particular ISIS database and restart database synchronization with the neighbors.
- o clear-isis-adjacency : restart a particular set of ISIS adjacencies.

rpcs:

```

+---x clear-isis-adjacency
|   +--ro input
|       +--ro routing-instance-name      rt:routing-instance-state-ref
|       +--ro routing-protocol-instance-name  isis-instance-state-ref
|       +--ro isis-level?                  isis-level
|       +--ro interface?                   string
+---x clear-isis-database
    +--ro input
        +--ro routing-instance-name      rt:routing-instance-state-ref
        +--ro routing-protocol-instance-name  isis-instance-state-ref
        +--ro isis-level?                  isis-level

```

### 4. Notifications

The "ietf-isis" module a new notification "isis-adjacency-updown". This notification is triggered when an ISIS adjacency moves to Up or Down state.

This notification provides details on the affected adjacency and its new state.

notifications:

```
+---n isis-adjacency-updown
  +---ro interface?          string
  +---ro neighbor?           string
  +---ro neighbor-system-id? isis-system-id
  +---ro isis-level?         isis-level
  +---ro state?              enumeration
  +---ro reason?             string
```

## 5. Interaction with other YANG modules

The "isis-cfg" container augments the "/rt:routing/rt:routing-instance/rt:routing-protocols/routing-protocol" container of the ietf-routing module by defining ISIS specific parameters.

The "isis-state" container augments the "/rt:routing-state/rt:routing-instance/rt:routing-protocols/routing-protocol" container of the ietf-routing module by defining ISIS specific operational states.

Some ISIS specific routes attributes are added to route objects of the ietf-routing module by augmenting "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" and "/rt:active-route/rt:output/rt:route".

## 6. Yang module

<CODE BEGINS> file "ietf-isis@2014-06-25.yang"

```
module ietf-isis {
  namespace "urn:ietf:params:xml:ns:yang:ietf-isis";

  prefix isis;

  import ietf-routing {
    prefix "rt";
  }

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF ISIS Working Group";
```

```
contact
  "WG List:    <mailto:isis-wg@ietf.org>

  Editor:      Stephane Litkowski
                <mailto:stephane.litkowski@orange.com>";

description
  "The YANG module defines a generic configuration model for
  ISIS common across all of the vendor implementations.";

revision 2014-06-25 {
  description "
    * isis-cfg renamed to isis.
    * Add precisions on authentication-keys in description
    ";
  reference "draft-litkowski-isis-yang-isis-01";
}

revision 2014-06-20 {
  description "
    * isis-op renamed to isis-state.
    * Multiple instances under isis are removed.
    * interface-cfg grouping removed and content
      is directly included in container isis.
    * TLVxx renamed with human-readable name in isis-database.
      TLV reference are putted in description.
    * Reference to core routing module were fixed.
    * Namespace fixed.
    * Add simple-iso-address type.
    * area-id and system-id in isis container are merged to
      nsap-address.
    * Add isis-system-id type.
    * Add isis-lsp-id type.
    * Add remaining-lifetime leaf in isis-database.
    * Add TLV2 (is-neighbor) in isis-database.
    * Renamed some container name for consistency
      reason ('isis-' prefixed).
    * Add new identities isis-cfg and isis-state.
    * Add descriptions.
    * Add notification isis-adjacency-updown.
    * Add RPC clear-isis-adjacency and clear-isis-database.
    ";
  reference "draft-litkowski-isis-yang-isis-00";
}

revision 2014-06-11 {
  description "Initial revision.";
```

```
    reference "draft-litkowski-netmod-isis-cfg-00";
}
identity isis {
    base rt:routing-protocol;
    description "Identity for the ISIS routing protocol.";
}

identity isis-state {
    description "Identity for the ISIS routing protocol
operational states.";
}

identity isis-adjacency-updown {
    description "Identity for the ISIS routing protocol
adjacency state.";
}

identity clear-isis-database {
    description "Identity for the ISIS routing protocol
database reset action.";
}

identity clear-isis-adjacency {
    description "Identity for the ISIS routing protocol
adjacency reset action.";
}

typedef isis-instance-state-ref {
    type leafref {
        path "/rt:routing-state/rt:routing-instance/"
        +"rt:routing-protocols/rt:routing-protocol/rt:name";
    }
    description
        "This type is used for leafs that reference state data of
an ISIS protocol instance.";
}

typedef isis-level {
    type enumeration {
        enum "level-1" {
            description
                "This enum describes L1 only capability.";
        }
        enum "level-2" {
            description
                "This enum describes L2 only capability.";
        }
    }
}
```

```
        enum "level-1-2" {
            description
                "This enum describes both level capability.";
        }
    }
    description
        "This type defines ISIS level of an object.";
}

typedef isis-lsp-id {
    type string {
        pattern
            '[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]'
            +'{4}\.[0-9][0-9]-[0-9][0-9]';
    }
    description
        "This type defines isis LSP ID using pattern,
        system id looks like : 0143.0438.AeF0.02-01";
}

typedef simple-iso-address {
    type string {
        pattern '[0-9A-Fa-f]{2}\.([0-9A-Fa-f]{4}\.){0,3}'
            +'[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.'
            +'[0-9][0-9]';
    }
    description
        "This type defines simple iso address format,
        it looks like : area_id.systemid.nsel
        The area ID is at least 1 byte of AFI, and is up to
        13 bytes.";
}

typedef isis-system-id {
    type string {
        pattern
            '[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.00';
    }
    description
        "This type defines isis system id using pattern,
        system id looks like : 0143.0438.AeF0.00";
}

typedef isis-wide-metric {
    type uint32 {
        range "0 .. 16777215";
    }
    description
```

```
        "This type defines wide style format
        of ISIS metric.";
    }

    typedef isis-std-metric {
        type uint8 {
            range "0 .. 63";
        }
        description
            "This type defines old style format
            of ISIS metric.";
    }

    grouping isis-route-content {
        description
            "This group add isis-specific route properties.";
        leaf metric {
            type uint32;
            description
                "This leaf describes isis metric of a route.";
        }
        leaf-list tag {
            type uint32;
            description
                "This leaf describes list of tags associated
                with the route.";
        }
        leaf route-type {
            type enumeration {
                enum l2-up-internal {
                    description "Level 2 internal route
                    and not leaked to a lower level";
                }
                enum l1-up-internal {
                    description "Level 1 internal route
                    and not leaked to a lower level";
                }
                enum l2-up-external {
                    description "Level 2 external route
                    and not leaked to a lower level";
                }
                enum l1-up-external {
                    description "Level 1 external route
                    and not leaked to a lower level";
                }
                enum l2-down-internal {
                    description "Level 2 internal route
                    and leaked to a lower level";
                }
            }
        }
    }
}
```

```
    }
    enum l1-down-internal {
        description "Level 1 internal route
            and leaked to a lower level";
    }
    enum l2-down-external {
        description "Level 2 external route
            and leaked to a lower level";
    }
    enum l1-down-external {
        description "Level 1 external route
            and leaked to a lower level";
    }
}
description
    "This leaf describes the type of isis route.";
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
    when "rt:source-protocol = 'isis:isis'" {
        description "ISIS-specific route attributes.";
    }
    uses isis-route-content;
    description
        "This augments route object in RIB with ISIS-specific
            attributes.";
}

augment "/rt:active-route/rt:output/rt:route"
{
    uses isis-route-content;
    description "ISIS-specific route attributes.";
}

grouping isis-prefix-ipv4-std {
    description
        "This group defines attributes of an
            IPv4 standard prefix.";
    leaf up-down {
        type boolean;
        description
            "This leaf expresses the value of up/down bit.";
    }
    leaf i-e {
        type boolean;
        description

```

```
        "This leaf expresses the value of I/E bit.";
    }
    leaf ip-prefix {
        type inet:ipv4-address;
        description
            "This leaf describes the IPv4 prefix";
    }
    leaf prefix-len {
        type uint8;
        description
            "This leaf describes the IPv4 prefix len in bits";
    }
    leaf default-metric {
        type isis-std-metric;
        description
            "This leaf describes the isis default metric value";
    }
    container delay-metric {
        leaf metric {
            type isis-std-metric;
            description
                "This leaf describes the isis delay metric value";
        }
        leaf supported {
            type boolean;
            default "false";
            description
                "This leaf describes if the metric is supported.";
        }
    }
    description
        "This container defines the ISIS delay metric.";
}
container expense-metric {
    leaf metric {
        type isis-std-metric;
        description
            "This leaf describes the isis delay metric value";
    }
    leaf supported {
        type boolean;
        default "false";
        description
            "This leaf describes if the metric is supported.";
    }
    description
        "This container defines the ISIS expense metric.";
}
```



```
    container error-metric {
        leaf metric {
            type isis-std-metric;
            description
                "This leaf describes the isis delay metric value";
        }
        leaf supported {
            type boolean;
            default "false";
            description
                "This leaf describes if the metric is supported.";
        }

        description
            "This container defines the ISIS error metric.";
    }
}

grouping isis-prefix-ipv4-extended {
    description
        "This group defines attributes of an
        IPv4 extended prefix.";
    leaf up-down {
        type boolean;
        description
            "This leaf expresses the value of up/down bit.";
    }
    leaf ip-prefix {
        type inet:ipv4-address;
        description
            "This leaf describes the IPv4 prefix";
    }
    leaf prefix-len {
        type uint8;
        description
            "This leaf describes the IPv4 prefix len in bits";
    }
}

leaf metric {
    type isis-wide-metric;
    description
        "This leaf describes the isis metric value";
}
leaf-list tag {
    type uint32;
    description
        "This leaf describes a list of tags associated with
        the prefix.";
```

```
    }
  }

  grouping isis-prefix-ipv6-extended {
    description
      "This group defines attributes of an
      IPv6 prefix.";
    leaf up-down {
      type boolean;
      description
        "This leaf expresses the value of up/down bit.";
    }
    leaf ip-prefix {
      type inet:ipv6-address;
      description
        "This leaf describes the IPv6 prefix";
    }
    leaf prefix-len {
      type uint8;
      description
        "This leaf describes the IPv4 prefix len in bits";
    }
    leaf metric {
      type isis-wide-metric;
      description
        "This leaf describes the isis metric value";
    }
    leaf-list tag {
      type uint32;
      description
        "This leaf describes a list of tags associated with
        the prefix.";
    }
  }

  grouping isis-neighbor-extended {
    description
      "This group defines attributes of an
      ISIS extended neighbor.";
    leaf neighbor-id {
      type isis-system-id;
      description
        "This leaf describes the system-id of the neighbor.";
    }
    leaf metric {
      type isis-wide-metric;
      description
```

```
        "This leaf describes the isis metric value";
    }
}

grouping isis-neighbor {
    description
        "This group defines attributes of an
        ISIS standard neighbor.";
    leaf neighbor-id {
        type isis-system-id;
        description
            "This leaf describes the system-id of the neighbor.";
    }
    leaf i-e {
        type boolean;
        description
            "This leaf expresses the value of I/E bit.";
    }
    leaf default-metric {
        type isis-std-metric;
        description
            "This leaf describes the isis default metric value";
    }
    container delay-metric {
        leaf metric {
            type isis-std-metric;
            description
                "This leaf describes the isis delay metric value";
        }
        leaf supported {
            type boolean;
            default "false";
            description
                "This leaf describes if the metric is supported.";
        }
        description
            "This container defines the ISIS delay metric.";
    }
    container expense-metric {
        leaf metric {
            type isis-std-metric;
            description
                "This leaf describes the isis delay metric value";
        }
        leaf supported {
            type boolean;
            default "false";
            description
```

```
        "This leaf describes if the metric is supported.";
    }
    description
        "This container defines the ISIS expense metric.";
}
container error-metric {
    leaf metric {
        type isis-std-metric;
        description
            "This leaf describes the isis delay metric value";
    }
    leaf supported {
        type boolean;
        default "false";
        description
            "This leaf describes if the metric is supported.";
    }
    description
        "This container defines the ISIS error metric.";
}
}

grouping isis-database {
    description
        "This group defines attributes of an
        ISIS database (Link State DB).";
    leaf lsp-id {
        type isis-lsp-id;
        description
            "This leaf describes the LSP ID of the LSP.";
    }
    leaf checksum {
        type uint16;
        description
            "This leaf describes the checksum of the LSP.";
    }
    leaf remaining-lifetime {
        type uint16;
        units "seconds";
        description
            "This leaf describes the remaining lifetime
            in seconds before the LSP expiration.";
    }
    leaf sequence {
        type uint32;
        description
            "This leaf describes the sequence number of the LSP.";
    }
}
```

```
leaf attributes {
  type bits {
    bit PARTITIONNED {
      description
        "If set, the originator supports partition
        repair.";
    }
    bit ATTACHED-ERROR {
      description
        "If set, the originator is attached to
        another area using the refered metric.";
    }
    bit ATTACHED-EXPENSE {
      description
        "If set, the originator is attached to
        another area using the refered metric.";
    }
    bit ATTACHED-DELAY {
      description
        "If set, the originator is attached to
        another area using the refered metric.";
    }
    bit ATTACHED-DEFAULT {
      description
        "If set, the originator is attached to
        another area using the refered metric.";
    }
    bit OVERLOAD {
      description
        "If set, the originator is overloaded,
        and must be avoided in path calculation.";
    }
  }
  description
    "This leaf describes attributes of the LSP.";
}

container is-neighbor {
  list neighbor {
    key "neighbor-id";
    uses isis-neighbor-extended;
    description
      "List of neighbors.";
  }
  description
    "This leaf describes list of ISIS extended neighbors.
    ISIS reference is TLV 2.";
}
```

```
container authentication {
  leaf authentication-type {
    type uint8;
    description
      "This leaf describes the authentication type
       to be used.";
  }
  leaf authentication-key {
    type string;
    description
      "This leaf describes the authentication key
       to be used. For security reason, the
       authentication key MUST NOT be presented
       in plaintext format. Authors recommends
       to use MD5 hash to present the authentication-key.";
  }
  description "This container describes authentication
  information of the node. ISIS reference is TLV 10.";
}

container extended-is-neighbor {
  list neighbor {
    key "neighbor-id";
    uses isis-neighbor-extended;
    description
      "List of neighbors.";
  }
  description
    "This container describes list of ISIS extended
    neighbors.
    ISIS reference is TLV 22.";
}

container ipv4-internal-reachability {
  list prefixes {
    key "ip-prefix";
    uses isis-prefix-ipv4-std;
    description
      "List of prefixes.";
  }
  description
    "This container describes list of ipv4 internal
    reachability information.
    ISIS reference is TLV 128.";
}

leaf-list protocol-supported {
  type uint8;
```

```
    description
      "This leaf describes the list of
      supported protocols.
      ISIS reference is TLV 129.";
  }

  container ipv4-external-reachability {
    list prefixes {
      key "ip-prefix";
      uses isis-prefix-ipv4-std;
      description
        "List of prefixes.";
    }
    description
      "This container describes list of ipv4 external
      reachability information.
      ISIS reference is TLV 130.";
  }

  leaf-list ipv4-addresses {
    type inet:ipv4-address;
    description
      "This leaf describes the ipv4 addresses of the node.
      ISIS reference is TLV 132.";
  }

  leaf ipv4-te-routerid {
    type inet:ipv4-address;
    description
      "This leaf describes the IPv4 Traffic Engineering
      router ID of the node.
      ISIS reference is TLV 134.";
  }

  container extended-ipv4-reachability {
    list prefixes {
      key "ip-prefix";
      uses isis-prefix-ipv4-extended;
      description
        "List of prefixes.";
    }
    description
      "This container describes list of ipv4 extended
      reachability information.
      ISIS reference is TLV 135.";
  }
```

```
leaf dynamic-hostname {
    type string;

    description
        "This leaf describes the name of the node.
        ISIS reference is TLV 137.";
}

leaf ipv6-te-routerid {
    type inet:ipv6-address;
    description
        "This leaf describes the IPv6 Traffic Engineering
        router ID of the node.
        ISIS reference is TLV 140.";
}

container mt-is-neighbor {
    list neighbor {
        key "neighbor-id";
        leaf MT-ID {
            type uint16 {
                range "0 .. 4095";
            }
            description
                "This leaf defines the identifier
                of a topology.";
        }
        uses isis-neighbor-extended;
        description
            "List of neighbors.";
    }
    description
        "This container describes list of ISIS multi-topology
        neighbors.
        ISIS reference is TLV 223.";
}

container mt-entries {
    list topology {
        key "MT-ID";

        leaf MT-ID {
            type uint16 {
                range "0 .. 4095";
            }
            description
                "This leaf defines the identifier
```



```
        of a topology.";
    }

    leaf attributes {
        type bits {
            bit OVERLOAD {
                description
                "If set, the originator is overloaded,
                and must be avoided in path
                calculation.";
            }
            bit ATTACHED {
                description
                "If set, the originator is attached to
                another area using the refered metric.";
            }
        }
        description
        "This leaf describes attributes of the LSP
        for the associated topology.";
    }
    description
    "List of topologies supported.";
}
description
"This container describes the topology supported.
ISIS reference is TLV 229.";
}

leaf-list ipv6-addresses {
    type inet:ipv6-address;
    description
    "This leaf describes the ipv6 interface
    addresses of the node.
    ISIS reference is TLV 232.";
}

container mt-extended-ipv4-reachability {
    list prefixes {
        key "ip-prefix";
        leaf MT-ID {
            type uint16 {
                range "0 .. 4095";
            }
            description

```

```
        "This leaf defines the identifier
          of a topology.";
    }
    uses isis-prefix-ipv4-extended;
    description
      "List of prefixes.";
  }
  description
    "This container describes list of ipv4
    reachability information in multi-topology
    environment.
    ISIS reference is TLV 235.";
}

container mt-ipv6-reachability {
  list prefixes {
    key "ip-prefix";
    leaf MT-ID {
      type uint16 {
        range "0 .. 4095";
      }
      description
        "This leaf defines the identifier
        of a topology.";
    }
    uses isis-prefix-ipv6-extended;
    description
      "List of prefixes.";
  }
  description
    "This container describes list of ipv6
    reachability information in multi-topology
    environment.
    ISIS reference is TLV 237.";
}

container ipv6-reachability {
  list prefixes {
    key "ip-prefix";
    uses isis-prefix-ipv6-extended;
    description
      "List of prefixes.";
  }
  description
    "This container describes list of ipv6
    reachability information.
    ISIS reference is TLV 236.";
```

```
    }

    container router-capabilities {
        leaf binary {
            type binary;
            description
                "This leaf describes the capability of the node.
                Format is binary according to the protocol encoding.";
        }
        description
            "This container describes the capabilities of the node.
            This container may be extended with detailed
            information.
            ISIS reference is TLV 242.";
    }
}

grouping isis-address-family-cfg {
    description
        "This group defines address-family-cfg
        global configuration for ISIS.";
    leaf ipv4-unicast {
        type boolean;
        description
            "This leaf defines if IPv4 unicast is activated.";
    }
    leaf ipv6-unicast {
        type boolean;
        description
            "This leaf defines if IPv6 unicast is activated.";
    }
    leaf ipv4-multicast {
        type boolean;
        description
            "This leaf defines if IPv4 multicast is activated.";
    }
    leaf ipv6-multicast {
        type boolean;
        description
            "This leaf defines if IPv6 multicast is activated.";
    }
}

grouping isis-interface-hello-cfg {
    description
        "This group defines hello interface
        parameters for ISIS.";
    leaf hello-authentication-type {
```

```
    type enumeration {
      enum none {
        description "No authentication used.";
      }
      enum plaintext {
        description "Plain text password used.";
      }
      enum message-digest {
        description "MD5 digest used.";
      }
    }
    description
      "This leaf describes the authentication type
      to be used in hello messages.";
  }
  leaf hello-authentication-key {
    type string;
    description
      "This leaf describes the authentication key
      to be used in hello messages.
      For security reason, the
      authentication key MUST NOT be presented
      in plaintext format upon a get-config reply.
      Authors recommends
      to use MD5 hash to present the authentication-key";
  }
  leaf hello-interval {
    type uint16;
    units "seconds";
    description
      "This leaf defines the interval of hello messages.";
  }
  leaf hello-multiplier {
    type uint16;
    description
      "This leaf defines the number of hello failed to be
      received before declaring the adjacency down.";
  }
}

grouping isis-interface-level-cfg {
  description
    "This group defines level specific
    configuration for ISIS interfaces.";
  uses isis-interface-hello-cfg;

  uses isis-address-family-cfg;
```

```
    leaf priority {
      type uint8 {
        range "0 .. 127";
      }

      description
        "This leaf describes the priority of the interface
        for DIS election.";
    }
    leaf ipv4-unicast-metric {
      type isis-wide-metric;
      description
        "This leaf describes the IPv4 unicast metric
        of the interface.";
    }
    leaf ipv6-unicast-metric {
      type isis-wide-metric;
      description
        "This leaf describes the IPv6 unicast metric
        of the interface.";
    }
    leaf ipv4-multicast-metric {
      type isis-wide-metric;
      description
        "This leaf describes the IPv4 multicast metric
        of the interface.";
    }
    leaf ipv6-multicast-metric {
      type isis-wide-metric;
      description
        "This leaf describes the IPv6 multicast metric
        of the interface.";
    }
    leaf passive {
      type boolean;
      default "false";
      description
        "This leaf defines if interface is in passive mode
        (ISIS not running, but network is advertised).";
    }
  }
}

grouping isis-authentication-cfg {
  description
    "This group defines authentication
    configuration for ISIS.";
  leaf psnp-authentication {
    type boolean;
  }
}
```

```
        default "true";
        description
            "This leaf describes if PSNP messages must be
            authenticated.";
    }
    leaf csnp-authentication {
        type boolean;
        default "true";
        description
            "This leaf describes if CSNP messages must be
            authenticated.";
    }
    leaf hello-authentication {
        type boolean;
        default "true";
        description
            "This leaf describes if HELLO messages must be
            authenticated.";
    }
    leaf authentication-key {
        type string;
        description
            "This leaf describes the authentication key
            to be used.
            For security reason, the
            authentication key MUST NOT be presented
            in plaintext format upon a get-config reply.
            Authors recommends
            to use MD5 hash to present the authentication-key";
    }
    leaf authentication-type {
        type enumeration {
            enum none {
                description "No authentication used.";
            }
            enum plaintext {
                description "Plain text password used.";
            }
            enum message-digest {
                description "MD5 digest used.";
            }
        }
        description
            "This leaf describes the authentication type
            to be used.";
    }
}
```

```
grouping isis-level-cfg {
  description
    "This group defines level specific
    global configuration for ISIS.";
  leaf enabled {
    type boolean;
    default "true";
    description
      "This leaf defines the status of the administrative
      status of the level (Active / not active).";
  }

  uses isis-authentication-cfg;

  leaf metric-type {
    type enumeration {
      enum wide-only {
        description
          "Advertise new metric style only (RFC5305)";
      }
      enum old-only {
        description
          "Advertise old metric style only (RFC1195)";
      }
      enum both {
        description "Advertise both metric styles";
      }
    }
    description
      "This leaf describes the type of metric to be generated.
      Wide-only means only new metric style is generated,
      old-only means that only old style metric is generated,
      and both means that both are advertised.";
  }
  leaf preference {
    type uint8;
    description
      "This leaf defines the protocol preference.";
  }
  leaf external-preference {
    type uint8;
    description
      "This leaf defines the protocol preference for external
      routes.";
  }
  leaf default-ipv4-unicast-metric {
    type isis-wide-metric;
    description
```

```
        "This leaf defines the IPv4 unicast default metric.";
    }
    leaf default-ipv6-unicast-metric {
        type isis-wide-metric;
        description
            "This leaf defines the IPv6 unicast default metric.";
    }
    leaf default-ipv4-multicast-metric {
        type isis-wide-metric;
        description
            "This leaf defines the IPv4 multicast default metric.";
    }
    leaf default-ipv6-multicast-metric {
        type isis-wide-metric;
        description
            "This leaf defines the IPv6 multicast default metric.";
    }
}

augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
+ "rt:routing-protocol" {
    when "rt:type = 'isis:isis'" {
        description
            "This augment is only valid when routing protocol
            instance type is isis.";
    }
    description
        "This augments a routing protocol instance with ISIS
        specific parameters.";
    container isis {
        leaf isis-level {
            type isis-level;
            default "level-1-2";
            description
                "This leaf describes the type of ISIS node.
                A node can be level-1-only, level-2-only
                or level-1-2.
                ";
        }
        leaf nsap-address {
            type simple-iso-address;
            description
                "This leaf defines the NSAP address of the node
                in a simple format. This parameter is mandatory.";
        }
        leaf ipv4-router-id {
            type inet:ipv4-address;
            description
```



```
        "Router ID value that would be used in TLV134.";
    }
    leaf ipv6-router-id {
        type inet:ipv6-address;
        description
            "Router ID value that would be used in TLV234.";
    }
    leaf reference-bandwidth {
        type uint32;
        units "bps";
        description
            "This leaf defines the bandwidth for calculating
            metric.";
    }

    leaf lsp-mtu {
        type uint16;
        units "bytes";
        default 1492;
        description
            "This leaf describes the maximum size of a
            LSP PDU in bytes.";
    }
    leaf lsp-lifetime {
        type uint16;
        units "seconds";
        description
            "This leaf describes the lifetime of the router
            LSP in seconds.";
    }
    leaf lsp-refresh {
        type uint16;
        units "seconds";
        description
            "This leaf describes the refresh interval of the
            router LSP in seconds.";
    }
}

uses isis-authentication-cfg;

container isis-multi-topology-cfg {
    uses isis-address-family-cfg;
    description
        "This container describes activation of MT extensions
        for supporting new address families.";
}

container isis-level-1-cfg {
```

```
    uses isis-level-cfg;
    description
        "Defines configuration parameters of level 1.";
}

container isis-level-2-cfg {
    uses isis-level-cfg;
    description
        "Defines configuration parameters of level 2.";
}

container overload {
    leaf status {
        type boolean;
        description
            "This leaf defines the overload status.";
    }
    leaf timeout {
        type uint16;
        units "seconds";
        description
            "This leaf defines the timeout in seconds
            of the overload condition.";
    }
    description
        "This leaf describes if the router is
        set to overload state.";
}

container interfaces {
    list interface {
        key "name";
        leaf name {
            type leafref {
                path "/rt:routing/rt:routing-instance/"
                    +"rt:interfaces/rt:interface/rt:name";
            }
            description
                "Reference to the interface within
                the routing-instance.";
        }
        leaf level {
            type isis-level;
            default "level-1-2";
            description
                "This leaf defines the associated ISIS level
                of the interface.";
        }
    }
}
```

```
}
leaf lsp-interval {
    type uint16;
    units "milliseconds";
    description
        "This leaf defines the interval between LSP
        transmissions in msec";
}
leaf passive {
    type boolean;
    default "false";
    description
        "This leaf defines if interface is in passive
        mode (ISIS not running,
        but network is advertised).";
}
leaf csnp-interval {
    type uint16;
    units "seconds";
    description
        "This leaf defines the interval of CSNP
        messages.";
}

uses isis-interface-hello-cfg;

leaf hello-padding {
    type boolean;
    description
        "This leaf defines if ISIS Hellos would be
        padded up to MTU size.";
}

uses isis-address-family-cfg;

leaf interface-type {
    type enumeration {
        enum broadcast {
            description "Broadcast interface type.
            Would result in DIS election.";
        }
        enum point-to-point {
            description
                "Point to point interface type.";
        }
    }
    description
```

```
        "This leaf defines the type of adjacency
        to be established on the interface.
        This is affecting the type of hello
        message that would be used.";
    }

    leaf enabled {
        type boolean;
        default "true";
        description
            "This leaf describes the administrative
            status of the ISIS interface.";
    }

    leaf-list tag {
        type uint32;
        description
            "This leaf defines list of tags associated
            with the interface.";
    }

    container level-1 {
        uses isis-interface-level-cfg;
        description
            "This container defines the level 1 specific
            configuration of the interface.";
    }

    container level-2 {
        uses isis-interface-level-cfg;
        description
            "This container defines the level 2 specific
            configuration of the interface.";
    }

    description
        "List of ISIS interfaces.";
}
description
    "This container defines ISIS interface specific
    configuration objects.";
}

description
    "This container defines ISIS specific configuration
    objects.";
```

```
}

augment "/rt:routing-state/rt:routing-instance/"
  + "rt:routing-protocols/rt:routing-protocol" {
    when "rt:type = 'isis:isis'" {
      description
        "This augment is only valid when routing protocol
        instance type is isis.";
    }
  }
description
  "This augments routing protocol instance states with ISIS
  specific parameters.";
container isis-state {
  config false;
  container adjacencies {

    list adjacency {
      key interface;

      leaf interface {
        type string;
        description
          "This leaf describes the name
          of the interface.";
      }
      leaf level {
        type uint8 {
          range "1 .. 2";
        }
        description
          "This leaf describes the associated
          ISIS level of the interface.
          The value of the level can only be 1
          or 2.";
      }
    }
    leaf state {
      type enumeration {
        enum "Up" {
          description
            "This state describes that
            adjacency is established.";
        }
        enum "Down" {
          description
            "This state describes that
            adjacency is NOT established.";
        }
        enum "Init" {
```

```
        description
        "This state describes that
        adjacency is establishing.";
    }
}
description
    "This leaf describes the state of the
    interface.";
}
description
    "List of operational adjacencies.";
}
description
    "This container lists the adjacencies of
    the local node.";
}
container spf-log {
    list event {
        key id;

        leaf id {
            type uint32;
            description
                "This leaf defines the event identifier.
                This is a purely internal value.";
        }
        leaf spf-type {
            type enumeration {
                enum full {
                    description
                        "Computation done is a Full SPF.";
                }
                enum incremental {
                    description
                        "Computation done is an
                        incremental SPF.";
                }
                enum route-only {
                    description
                        "Computation done is a
                        reachability computation
                        only.";
                }
            }
        }
        description
            "This leaf describes the type of computation
            used.";
    }
}
```

```
leaf level {
  type uint8 {
    range "1 .. 2";
  }
  description
    "This leaf describes the level affected bytes
    the computation.";
}
leaf spf-delay {
  type uint32;
  units "milliseconds";
  description
    "This leaf describes the SPF delay that
    was used for this event.";
}
leaf schedule-timestamp {
  type yang:timestamp;
  description
    "This leaf describes the timestamp
    when the computation was scheduled.";
}
leaf start-timestamp {
  type yang:timestamp;
  description
    "This leaf describes the timestamp
    when the computation was started.";
}
leaf end-timestamp {
  type yang:timestamp;
  description
    "This leaf describes the timestamp
    when the computation was ended.";
}
list trigger-lsp {
  key "lsp";
  leaf lsp {
    type isis-lsp-id;
    description
      "This leaf describes the LSPID
      of the LSP.";
  }
  leaf sequence {
    type uint32;
    description
      "This leaf describes the sequence
      number of the LSP.";
  }
  description
```

```
        "This leaf describes list of LSPs
          that triggered the computation.";
    }
    description
    "List of computation events.";
}

description
    "This container lists the SPF computation events.";
}
container lsp-log {
    list event {
        key id;

        leaf id {
            type uint32;
            description
            "This leaf defines the event identifier.
              This is a purely internal value.";
        }
        leaf level {
            type uint8 {
                range "1 .. 2";
            }
            description
            "This leaf describes the level affected bytes
              the computation.";
        }
    }
    container lsp {
        leaf lsp {

            type isis-lsp-id;
            description
            "This leaf describes the LSPID
              of the LSP.";
        }
        leaf sequence {
            type uint32;
            description
            "This leaf describes the sequence
              number of the LSP.";
        }
    }
    description
    "This container describes the received LSP
      , in case of local LSP update the local
      LSP ID is referenced.";
}
```



```
    leaf received-timestamp {
        type yang:timestamp;

        description
            "This leaf describes the timestamp
            when the LSP was received. In case of
            local LSP update, the timestamp refers
            to the local LSP update time.";
    }

    description
        "List of LSP events.";
}

description
    "This container lists the LSP reception events.
    Local LSP modification are also contained in the
    list.";
}
container database {
    container level-1 {
        list lsp {
            key lsp-id;

            uses isis-database;
            description
                "List of LSPs in LSDB.";
        }

        description
            "This container describes the list of LSPs
            in the level-1 database.";
    }
    container level-2 {
        list lsp {
            key lsp-id;

            uses isis-database;
            description
                "List of LSPs in LSDB.";
        }

        description
            "This container describes the list of LSPs
            in the level-2 database.";
    }
}
description
    "This container describes ISIS Link State
```

```
        databases.";
    }
    container hostnames {

        list hostname {
            key system-id;
            leaf system-id {
                type isis-system-id;
                description
                "This leaf describes the system-id
                associated with the hostname.";
            }
            leaf hostname {

                type string;
                description
                "This leaf describes the hostname
                associated with the system ID.";
            }
            description
            "List of system-id/hostname associations";
        }

        description
        "This container describes the list
        of binding between system-id and
        hostnames.";
    }

    description
    "This container defines various ISIS states objects.";
}

/* RPC methods */

rpc clear-isis-adjacency {
    description
    "This RPC request clears a particular
    set of ISIS adjacencies. If the operation
    fails for ISIS internal reason, then
    error-tag and error-app-tag should be set
    to a meaningful value.";
    input {
        leaf routing-instance-name {
            type rt:routing-instance-state-ref;
            mandatory "true";
            description
```

"Name of the routing instance whose ISIS information 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'.";

```
}
leaf routing-protocol-instance-name {
  type isis-instance-state-ref;
  mandatory "true";
  description
    "Name of the ISIS protocol instance whose ISIS
     information is being queried.

    If the ISIS 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-protocol-instance-not-found'.";
}
leaf isis-level {
  type isis-level;
  description
    "ISIS level of the adjacency to be cleared.
    If ISIS level is level-1-2, both level 1 and level 2
    adjacencies would be cleared.

    If the value provided is different from the one
    authorized in the enum type, then this
    operation SHALL fail with error-tag 'data-missing'
    and error-app-tag
    'bad-isis-level'.
    ";
}
leaf interface {
  type string;
  description
    "Name of the ISIS interface.

    If the ISIS interface 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
    'isis-interface-not-found'.";
}
```

```
    }  
  }  
  
  rpc clear-isis-database {  
    description  
      "This RPC request clears a particular  
      ISIS database. If the operation  
      fails for ISIS internal reason, then  
      error-tag and error-app-tag should be set  
      to a meaningful value.";  
    input {  
      leaf routing-instance-name {  
        type rt:routing-instance-state-ref;  
        mandatory "true";  
        description  
          "Name of the routing instance whose ISIS  
          information 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'. ";  
      }  
      leaf routing-protocol-instance-name {  
        type isis-instance-state-ref;  
        mandatory "true";  
        description  
          "Name of the ISIS protocol instance whose ISIS  
          information is being queried.  
  
          If the ISIS 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-protocol-instance-not-found'. ";  
      }  
      leaf isis-level {  
        type isis-level;  
        description  
          "ISIS level of the adjacency to be cleared.  
          If ISIS level is level-1-2, both level 1 and level 2  
          adjacencies would be cleared.  
  
          If the value provided is different from the one  
          authorized in the enum type, then this  
          operation SHALL fail with error-tag 'data-missing'
```

```
        and error-app-tag
        'bad-isis-level'.
    ";
}
}

/* Notifications */

notification isis-adjacency-updown {
    leaf interface {
        type string;
        description
            "Describes the interface of the adjacency";
    }
    leaf neighbor {
        type string;
        description
            "Describes the name of the neighbor. If the
            name of the neighbor is not available, the
            field would be empty.";
    }
    leaf neighbor-system-id {
        type isis-system-id;
        description
            "Describes the system-id of the neighbor.";
    }
    leaf isis-level {
        type isis-level;
        description
            "Describes the ISIS level of the adjacency.";
    }
    leaf state {
        type enumeration {
            enum "Up" {
                description
                    "This state describes that
                    adjacency is established.";
            }
            enum "Down" {
                description
                    "This state describes that
                    adjacency is no more established.";
            }
        }
    }
}
```

```
        description
        "This leaf describes the new state of the
        ISIS adjacency.";
    }
    leaf reason {
        type string;
        description
        "If the adjacency is going to DOWN,
        this leaf provides a reason for the adjacency
        going down. The reason is provided as a text.
        If the adjacency is going to UP, no reason is
        provided.";
    }
    description
    "This notification is sent when an ISIS adjacency
    moves to Up state or to Down state.";
}

}

<CODE ENDS>
```

## 7. Security Considerations

Configuration and state data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241].

As ISIS is an IGP protocol (critical piece of the network), ensuring stability and security of the protocol is mandatory for the network service.

Authors recommends to implement NETCONF access control model ([RFC6536]) to restrict access to all or part of the configuration to specific users. Access control to RPCs is also critical as RPC permits to clear protocol datastructures that would definitively impact the network service. This kind of RPC needs only to be used in specific cases by well-known experienced users.

Authors consider that all the configuration is considered as sensitive/vulnerable as well as RPCs. But security teams can decide to open some part of the configuration to less experienced users depending on the internal organization, for example :

- o User FullWrite : would access to the whole data model. This kind of profile may be restricted to few experienced people.

- o User PartialWrite : would only access to configuration part within /isis/interfaces/interface. So this kind of profile is restricted to creation/modification/deletion of interfaces. This profile does not have access to RPC.
- o User Read : would only access to state part /isis-state.

Unauthorized access to configuration or RPC may cause high damages to the network service.

The /isis-state/database may contain authentication information. As presented in the description of the /isis-state/database/level-1/lsp/authentication/authentication-key, the authentication MUST never be presented in plaintext format for security reason. Authors recommends the usage of MD5 to present the authentication-key.

Some authentication-key may also be present in the /isis configuration. When configuring ISIS using the NETCONF protocol, authors recommends the usage of secure transport of NETCONF using SSH ([RFC6242]).

## 8. Acknowledgements

## 9. IANA Considerations

## 10. Normative References

- [I-D.ietf-netmod-routing-cfg]  
Lhotka, L., "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-15 (work in progress), May 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [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.

#### Appendix A. Example: NETCONF <get> Reply

This section gives an example of a reply to the NETCONF <get> request for a device that implements the data model defined in this document. The example is written in XML.

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:isis="urn:ietf:params:xml:ns:yang:ietf-isis"
  xmlns:v4ur="urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing"
  xmlns:rt="urn:ietf:params:xml:ns:yang:ietf-routing"
  xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ip="urn:ietf:params:xml:ns:yang:ietf-ip">
  <if:interfaces>
    <if:interface>
      <if:name>eth0</if:name>
      <ip:ipv4>
        <ip:address>
          <ip:ip>10.0.0.1</ip:ip>
          <ip:prefix-length>30</ip:prefix-length>
        </ip:address>
      </ip:ipv4>
    </if:interface>
    <if:interface>
      <if:name>lo0</if:name>
      <ip:ipv4>
        <ip:address>
          <ip:ip>192.168.0.2</ip:ip>
          <ip:prefix-length>32</ip:prefix-length>
        </ip:address>
      </ip:ipv4>
    </if:interface>
  </if:interfaces>
  <if:interfaces-state>
    <if:interface>
      <if:name>eth0</if:name>
      <if:oper-status>up</if:oper-status>
      <if:statistics>
        <if:discontinuity-time>
          2014-12-02T12:14:25.123Z
        </if:discontinuity-time>
        <if:in-octets>100</if:in-octets>
        <if:in-unicast-pkts>10</if:in-unicast-pkts>
        <if:in-broadcast-pkts>0</if:in-broadcast-pkts>
      </if:statistics>
    </if:interface>
  </if:interfaces-state>
</data>
```



```
<if:in-multicast-pkts>0</if:in-multicast-pkts>
<if:in-discards>0</if:in-discards>
<if:in-errors>0</if:in-errors>
<if:in-unknown-protos>0</if:in-unknown-protos>
<if:out-octets>541</if:out-octets>
<if:out-unicast-pkts>100</if:out-unicast-pkts>
<if:out-broadcast-pkts>0</if:out-broadcast-pkts>
<if:out-multicast-pkts>0</if:out-multicast-pkts>
<if:out-discards>0</if:out-discards>
<if:out-errors>0</if:out-errors>
</if:statistics>
<ip:ipv4>
  <ip:address>
    <ip:ip>10.0.0.1</ip:ip>
    <ip:prefix-length>30</ip:prefix-length>
  </ip:address>
</ip:ipv4>
</if:interface>
<if:interface>
  <if:name>lo0</if:name>
  <if:oper-status>up</if:oper-status>
  <if:statistics>
    <if:discontinuity-time>
      2014-12-02T12:14:25.123Z
    </if:discontinuity-time>
    <if:in-octets>100</if:in-octets>
    <if:in-unicast-pkts>10</if:in-unicast-pkts>
    <if:in-broadcast-pkts>0</if:in-broadcast-pkts>
    <if:in-multicast-pkts>0</if:in-multicast-pkts>
    <if:in-discards>0</if:in-discards>
    <if:in-errors>0</if:in-errors>
    <if:in-unknown-protos>0</if:in-unknown-protos>
    <if:out-octets>541</if:out-octets>
    <if:out-unicast-pkts>100</if:out-unicast-pkts>
    <if:out-broadcast-pkts>0</if:out-broadcast-pkts>
    <if:out-multicast-pkts>0</if:out-multicast-pkts>
    <if:out-discards>0</if:out-discards>
    <if:out-errors>0</if:out-errors>
  </if:statistics>
  <ip:ipv4>
    <ip:address>
      <ip:ip>192.168.0.2</ip:ip>
      <ip:prefix-length>32</ip:prefix-length>
    </ip:address>
  </ip:ipv4>
</if:interface>
</if:interfaces-state>
<rt:routing-state>
```

```
<rt:routing-instance>
  <rt:name>rtrl</rt:name>
  <rt:id>1</rt:id>
  <rt:router-id>192.0.2.1</rt:router-id>
  <rt:interfaces>
    <rt:interface>
      <rt:name>eth0</rt:name>
    </rt:interface>
  </rt:interfaces>
  <rt:routing-protocols>
    <rt:routing-protocol>
      <rt:name>ISIS1</rt:name>
      <rt:type>isis:isis</rt:type>
      <isis:isis-state>
        <isis:adjacencies>
          <isis:adjacency>
            <isis:interface>eth0</isis:interface>
            <isis:level>2</isis:level>
            <isis:state>Up</isis:state>
          </isis:adjacency>
        </isis:adjacencies>
        <isis:spf-log>
          <isis:event>
            <isis:id>18979</isis:id>
            <isis:spf-type>full</isis:spf-type>
            <isis:level>2</isis:level>
            <isis:spf-delay>150</isis:spf-delay>
            <isis:schedule-timestamp>
              1403612245
            </isis:schedule-timestamp>
            <isis:start-timestamp>
              1403612399
            </isis:start-timestamp>
            <isis:end-timestamp>
              1403612420
            </isis:end-timestamp>
            <isis:trigger-lsp>
              <isis:lsp>0000.1245.1245.01-01</isis:lsp>
              <isis:sequence>125458</isis:sequence>
            </isis:trigger-lsp>
          </isis:event>
        </isis:spf-log>
        <isis:lsp-log>
          <isis:event>
            <isis:id>1245</isis:id>
            <isis:level>2</isis:level>
            <isis:lsp>
              <isis:lsp>0000.1245.1245.01-01</isis:lsp>
```

```
<isis:sequence>125458</isis:sequence>
</isis:lsp>
<isis:received-timestamp>
1403612420
</isis:received-timestamp>
</isis:event>
</isis:lsp-log>
<isis:database>
<isis:level-2>
<isis:lsp>
<isis:lsp-id>0000.1245.1245.01-01</isis:lsp-id>
<isis:checksum>1245</isis:checksum>
<isis:remaining-lifetime>
45
</isis:remaining-lifetime>
<isis:sequence>125458</isis:sequence>
<isis:extended-is-neighbor>
<isis:neighbor>
<isis:neighbor-id>
0000.1245.9999.00
</isis:neighbor-id>
<isis:metric>100</isis:metric>
</isis:neighbor>
</isis:extended-is-neighbor>

<isis:protocol-supported>
204
</isis:protocol-supported>

<isis:ipv4-addresses>
192.168.0.2
</isis:ipv4-addresses>
<isis:extended-ipv4-reachability>
<isis:prefixes>
<isis:ip-prefix>192.168.0.2</isis:ip-prefix>
<isis:up-down>false</isis:up-down>
<isis:prefix-len>32</isis:prefix-len>
<isis:metric>30490</isis:metric>
<isis:tag>200</isis:tag>
</isis:prefixes>
</isis:extended-ipv4-reachability>
<isis:dynamic-hostname>
rtrl
</isis:dynamic-hostname>

</isis:lsp>
</isis:level-2>
</isis:database>
```

```
<isis:hostnames>
  <isis:hostname>
    <isis:system-id>
      0000.1245.9999.00
    </isis:system-id>
    <isis:hostname>rtrl</isis:hostname>
  </isis:hostname>
</isis:hostnames>
</isis:isis-state>
</rt:routing-protocol>
</rt:routing-protocols>
</rt:routing-instance>
<rt:ribs>
  <rt:rib>
    <rt:name>ipv4-master</rt:name>
    <rt:id>1</rt:id>
    <rt:address-family>
      v4ur:ipv4-unicast
    </rt:address-family>
    <rt:routes>
      <rt:route>
        <rt:id>124554657</rt:id>
        <rt:outgoing-interface>eth0</rt:outgoing-interface>
        <rt:source-protocol>isis:isis</rt:source-protocol>
        <rt:last-updated>
          2013-07-02T18:02:45+01:00
        </rt:last-updated>
        <v4ur:destination-prefix>
          10.0.0.0/24
        </v4ur:destination-prefix>
        <isis:metric>750</isis:metric>
        <isis:route-type>l2-up-internal</isis:route-type>

      </rt:route>
    </rt:routes>

  </rt:rib>
</rt:ribs>

</rt:routing-state>
<rt:routing>
  <rt:routing-instance>
    <rt:name>rtrl</rt:name>
    <rt:router-id>192.0.2.1</rt:router-id>
    <rt:interfaces>
      <rt:interface>
        <rt:name>eth0</rt:name>
      </rt:interface>
```

```
<rt:interface>
  <rt:name>lo0</rt:name>
</rt:interface>
</rt:interfaces>
<rt:routing-protocols>
  <rt:routing-protocol>
    <rt:name>ISIS1</rt:name>
    <rt:type>isis:isis</rt:type>
    <isis:isis>
      <isis:isis-level>level-2</isis:isis-level>
      <isis:nsap-address>
        49.0002.0000.1245.9999.00
      </isis:nsap-address>
      <isis:lsp-lifetime>65535</isis:lsp-lifetime>
      <isis:lsp-refresh>65000</isis:lsp-refresh>
      <isis:authentication-key>
        f567acafcd6578861df5683268$
      </isis:authentication-key>
      <isis:authentication-type>
        plaintext
      </isis:authentication-type>
      <isis:isis-multi-topology-cfg>
        <isis:ipv4-unicast>true</isis:ipv4-unicast>
      </isis:isis-multi-topology-cfg>

      <isis:isis-level-2-cfg>
        <isis:metric-type>wide-only</isis:metric-type>
        <isis:default-ipv4-unicast-metric>
          11111111
        </isis:default-ipv4-unicast-metric>
      </isis:isis-level-2-cfg>
    <isis:interfaces>
      <isis:interface>
        <isis:name>eth0</isis:name>
        <isis:lsp-interval>100</isis:lsp-interval>
        <isis:csnp-interval>10</isis:csnp-interval>
        <isis:interface-type>
          point-to-point
        </isis:interface-type>
        <isis:level-2>
          <isis:ipv4-unicast-metric>
            200
          </isis:ipv4-unicast-metric>
        </isis:level-2>
      </isis:interface>
      <isis:interface>
        <isis:name>lo0</isis:name>
        <isis:level-2>
```

```
<isis:ipv4-unicast-metric>
1
</isis:ipv4-unicast-metric>
</isis:level-2>
<isis:passive>true</isis:passive>
</isis:interface>
</isis:interfaces>
</isis:isis>
</rt:routing-protocol>
</rt:routing-protocols>
</rt:routing-instance>
</rt:routing>
</data>
```

## Author's Address

Stephane Litkowski  
Orange

Email: [stephane.litkowski@orange.com](mailto:stephane.litkowski@orange.com)

NETMOD  
Internet Draft  
Intended status: Standards Track

Tissa Senevirathne  
Norman Finn  
Deepak Kumar  
Samer Salam  
Carlos Pignataro  
Cisco

June 10, 2014

Expires: December 2014

YANG Data Model for Generic Operations, Administration, and  
Maintenance (OAM)  
draft-tissa-netmod-oam-01.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on December 10, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document presents YANG Data model for OAM. It provides a protocol-independent and technology-independent abstraction of key OAM constructs. These abstractions span OAM configuration and operational data; they promote uniformity between OAM technologies and support nested OAM workflows (i.e., performing OAM functions at different layers through a unified interface).

## Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	4
2.1. Terminology.....	4
3. Architecture of OAM YANG Model.....	5
4. Overview of the OAM Model.....	6
4.1. Maintenance Domain (MD) configuration.....	7
4.2. Maintenance Association (MA) configuration.....	7
4.3. Maintenance Endpoint (MEP) configuration.....	8
4.4. rpc definitions.....	9
5. OAM data hierarchy.....	11
6. OAM YANG module.....	17
7. Base Mode for IP.....	33
7.1. MEP Address.....	33
7.2. MEP ID for Base Mode.....	34
7.3. Maintenance Domain.....	34
7.4. Maintenance Association.....	34
8. Security Considerations.....	34
9. IANA Considerations.....	35
10. References.....	35
10.1. Normative References.....	35
10.2. Informative References.....	35
11. Acknowledgments.....	36

## 1. Introduction

Operations, Administration, and Maintenance (OAM) are important networking functions that allow operators to:



1. Configure networks
2. Monitor networks (Connectivity Verification, Continuity Check)
3. Troubleshoot failures (Fault verification and isolation).

An overview of OAM tools is presented at [OAMOVW].

Ping and Traceroute [RFC792], [RFC4443] are well-known fault verification and isolation tools, respectively, for IP networks. Over the years different technologies have developed similar tools for similar purposes.

[8021Q] Connectivity Fault Management is a well-established OAM standard that is widely adopted for Ethernet networks. ITU-T [Y1731], MEF Service OAM, MPLS-TP [RFC6371], TRILL [TRILLOAMFM] all define OAM methods based on [8021Q] CFM.

Given the wide adoption of the underlying OAM concepts defined in [8021Q] CFM, it is a reasonable choice to develop the unified OAM framework based on those concepts. In this document, we take the [8021Q] CFM model and extend it to a technology independent framework and build the corresponding YANG model accordingly. The YANG model presented in this document is the base model and supports IP Ping and Traceroute. The generic OAM YANG model is designed such that it can be extended to cover various technologies. Technology dependent nodes and RPC commands are defined in technology specific YANG models, which use and extend the base model defined here. As an example, VXLAN uses source UDP port number for flow entropy, while MPLS [RFC4379] uses IP addresses or the label stack for flow entropy in the hashing for multipath selection. To capture this variation, corresponding YANG models would define the applicable structures as augmentation to the generic base model presented here. This accomplishes three purposes: first it keeps each YANG model smaller and manageable. Second, it allows independent development of corresponding YANG models. Third, implementations can limit support to only the applicable set of YANG models. (e.g. TRILL RBridge may only need to implement Generic OAM model and the TRILL YANG model).

All implementations that follow the YANG framework presented in this document MUST implement the generic OAM YANG model presented here.

The unification of OAM, according to the proposal of this document, occurs at the management layer. Encapsulations and state machines may differ according to each protocol. A user who wishes to issues a Ping command or a Traceroute or initiate a performance monitoring session

can do so in the same manner regardless of the underlying protocol or technology.

As an example, consider a scenario where an IP ping from device A to Device B failed. Between device A and B there are IEEE 802.1 bridges a,b and c. Let's assume a,b and c are using [8021Q] CFM. A user upon detecting the IP layer ping failure, may decide to drill down to the Ethernet layer and issue the corresponding fault verification (LBM) and fault isolation (LTM) tools, using the same API. This ability to go up and down to different layers for troubleshooting is referred to as "nested OAM workflow" and is a useful concept that leads to efficient network troubleshooting and maintenance. The OAM YANG model presented in this document facilitates that without needing changes to the underlying protocols.

## 2. Conventions used in this document

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

### 2.1. Terminology

CCM - Continuity Check Message [8021Q]

ECMP - Equal Cost Multipath

LBM - Loopback Message [8021Q]

MP - Maintenance Point [8021Q]

MEP - Maintenance End Point [RFC7174] [8021Q] [RFC6371]

MIP - Maintenance Intermediate Point [RFC7174] [8021Q] [RFC6371]

MA - Maintenance Association [8021Q] [RFC7174]

MD - Maintenance Domain [8021Q]

MTV - Multi-destination Tree Verification Message

OAM - Operations, Administration, and Maintenance [RFC6291]

## TRILL - Transparent Interconnection of Lots of Links [RFC6325]

## 3. Architecture of OAM YANG Model

In this document we define the YANG model for Generic OAM. The YANG model defined here is generic such that other technologies can extend it for technology specific needs. The Generic OAM YANG model acts as the root for other OAM YANG models. This allows users to traverse between OAM of different technologies at ease through a uniform API set. This is also provides a nested OAM workflow. Figure 1 depicts the relationship of different OAM YANG models to the Generic OAM YANG Model. Some technologies may have different sub-technologies. As an example, consider Network Virtualization Overlays. These could employ either vXLAN or NVGRE as encapsulation. The Generic OAM YANG model provides a framework where technology-specific YANG models can inherit constructs from parent YANG models without needing to redefine them within the sub-technology.

Figure 1 depicts relationship of different YANG modules.

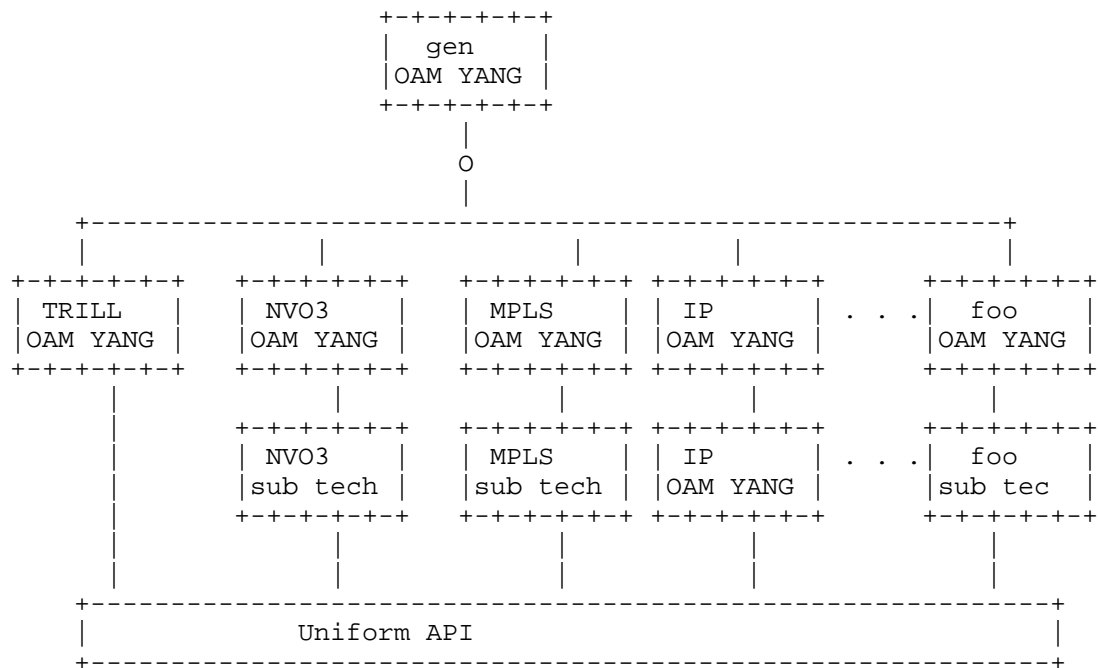


Figure 1 Relationship of TRILL OAM YANG model to generic YANG model

#### 4. Overview of the OAM Model

In this document we adopt the concepts of the [8021Q] CFM model and structure it such that it can be adapted to different technologies.

At the top of the Model is the Maintenance Domain. Each Maintenance Domain is associated with a Maintenance Name and a Domain Level.

Under each Maintenance Domain there is one or more Maintenance Association (MA). In IP, the MA can be per IP Subnet, in NVO3 this can be per VNI and for TRILL this can be per Fine-Grained Label or for VPLS this can be per VPLS instance.

Under each MA, there can be two or more MEPs (Maintenance End Points). MEPs are addressed by their respective technology specific address identifiers. The YANG model presented here provides flexibility to accommodate different addressing schemes.

In a parallel vertical, presented are the commands. Those, in YANG terms, are the rpc commands. These rpc commands provide uniform APIs for ping, traceroute and their equivalents as well as other OAM commands.

[8021Q] CFM framework requires explicit configuration of OAM entities prior to using any of the OAM tools. Users of Ping and Traceroute tools within IP devices are expecting ability to use OAM tools with no explicit configuration. In order to facilitate zero-touch experience, this document defines a default mode of OAM. The default mode of OAM is referred to as the Base Mode and specifies default values for each of the [8021Q] CFM parameters, such as Maintenance Domain Level, Name of the Maintenance Association and Addresses of MEP and so on. The default values of these depend on the technology. Base Mode for TRILL is defined in [TRILLOAMFM]. Section X of this document specifies the Base mode for IP devices. Base mode for other technologies such as NVO3, MPLS and future extensions will be defined in their corresponding documents.

It is important to note that, no specific enhancements are needed in the YANG model to support Base Mode. Implementations that comply with this document, by default implement the data nodes of the applicable technology. Data nodes of the Base Mode are read-only nodes.

#### 4.1. Maintenance Domain (MD) configuration

The container "domains" is the top level container within the ietf-oam module. Within the container "domains", separate list is maintained per MD. The MD list uses the key MD-name for indexing.

```

module: ietf-oam
  +--rw domains
  |   +--rw domain* [md-name]
  |   |   +--rw technology          identityref
  |   |   +--rw md-name-format      MD-name-format
  |   |   +--rw md-name             binary
  |   |   +--rw md-level            int32
  |   .
  |   .

```

Figure 1 Snippet of data hierarchy related to OAM domains

#### 4.2. Maintenance Association (MA) configuration

Within a given Maintenance Domain there can be one or more Maintenance Associations (MA). MAs are represented as a list and indexed by the MA-name.

```

module: ietf-oam
  +--rw domains
  |   +--rw domain* [md-name]
  |   |   +--rw technology          identityref
  |   |   +--rw md-name-format      MD-name-format
  |   |   +--rw md-name             binary
  |   |   +--rw md-level            int32
  |   |   +--rw MAs!
  |   |   |   +--rw MA* [ma-name]
  |   |   |   |   +--rw ma-name-format      MA-name-format
  |   |   |   |   +--rw ma-name             binary
  |   |   .
  |   |   .

```

Figure 2 Snippet of data hierarchy related to Maintenance Associations (MA).

#### 4.3. Maintenance Endpoint (MEP) configuration

Within a given Maintenance Association (MA), there can be one or more Maintenance End Points (MEP). MEPs are represented as a list within the data hierarchy and indexed by the key MEP-id.

```

module: ietf-oam
  +--rw domains
  |   +--rw domain* [md-name]
  |   |   +--rw technology          identityref
  |   |   +--rw md-name-format      MD-name-format
  |   |   +--rw md-name             binary
  |   |   +--rw md-level            int32
  |   |   +--rw MAs!
  |   |   |   +--rw MA* [ma-name]
  |   |   |   |   +--rw ma-name-format      MA-name-format
  |   |   |   |   +--rw ma-name            binary
  |   |   |
  |   |   .
  |   |   .
  |   |
  |   +--rw MEP* [mep-id]
  |   |   +--rw mep-id              MEP-id
  |   |   +--rw mep-name?          string
  |   |   +--rw mep-direction      MEP-direction
  |   |   +--rw ccm-Tx-enable?     boolean
  |   |   +--rw (mep-address)?
  |   |   |   +--:(mac-address)
  |   |   |   |   +--rw mac-address?      yang:mac-address
  |   |   |   +--:(ipv4-address)
  |   |   |   |   +--rw ipv4-address?     inet:ipv4-address
  |   |   |   +--:(ipv6-address)
  |   |   |   |   +--rw ipv6-address?     inet:ipv6-address
  |   |   +--rw (context-id)?
  |   |   |   +--:(context-null)
  |   |
  |   .
  |   .
  |   .

```

Figure 3 Snippet of data hierarchy related to Maintenance Endpoint (MEP).

#### 4.4. rpc definitions

The rpc model facilitates issuing commands to a NETCONF server (in this case to the device that need to execute the OAM command) and obtain a response. rpc model defined here abstracts OAM specific commands in a technology independent manner.

There are several rpc commands defined for the purpose of OAM. In this section we present a snippet of the ping command for illustration purposes. Please refer to Section 4 for the complete data hierarchy and Section 5 for the YANG model.

```

module: ietf-oam
  +--rw domains
  |   +--rw Domain* [MA-domain-name]
  |   |   +--rw technology technology
  |   |   +--rw MA-domain-name-format int32
  |   |   +--rw MA-domain-name binary
  |   |   +--rw MD-level int32
  |   .
  |   .
  rpcs:
    +---x ping
    |   +--ro input
    |   |   +--ro technology identityref
    |   |   +--ro md-name-format MD-name-format
    |   |   +--ro md-name? binary
    |   |   +--ro md-level int32
    |   |   +--ro ma-name-format MA-name-format
    |   |   +--ro ma-name binary
    |   |   +--ro (context-id)?
    |   |   |   +--:(context-null)
    |   |   |   |   +--ro context-null? empty
    |   |   +--ro (flow-entropy)?
    |   |   |   +--:(flow-entropy-null)
    |   |   +--ro ttl? uint8
    |   .
    |   .
    |   |   +--ro source-mep
    |   |   |   +--ro (mep-address)?
    |   .
    |   .
    |   |   |   +--ro mep-id? MEP-id
    |   |   +--ro destination-mep
    |   |   |   +--ro (mep-address)?
    |   .
    |   .
    |   +--ro output
    |   |   +--ro tx-packet-count? oam-counter32
    |   |   +--ro rx-packet-count? oam-counter32
    |   |   +--ro min-delay? oam-counter32
    |   |   +--ro average-delay? oam-counter32
    |   |   +--ro max-delay? oam-counter32

```

Figure 4 Snippet of data hierarchy related to rpc call Ping



## 5. OAM data hierarchy

The complete data hierarchy related to the OAM YANG model is presented below. The following notations are used within the data tree and carry the meaning as below.

Each node is printed as:

<status> <flags> <name> <opts> <type>

<status> is one of:

- + for current
- x for deprecated
- o for obsolete

<flags> is one of:

- rw for configuration data
- ro for non-configuration data
- x for rpcs
- n for notifications

<name> is the name of the node

If the node is augmented into the tree from another module, its name is printed as <prefix>:<name>.

<opts> is one of:

- ? for an optional leaf or choice
- ! for a presence container
- \* for a leaf-list or list
- [<keys>] for a list's keys

<type> is the name of the type for leafs and leaf-lists

```

module: gen-oam
  +--rw domains
    +--rw domain* [md-name technology]
      +--rw technology          identityref
      +--rw md-name-format      MD-name-format
      +--rw md-name             binary
      +--rw md-level            int32
    +--rw MAs!
      +--rw MA* [ma-name]
        +--rw ma-name-format      MA-name-format
        +--rw ma-name             binary
        +--rw (context-id)?
          | +--:(context-null)
          |   +--rw context-null?      empty
        +--rw ccm-Interval?      CCM-Interval
        +--rw ccm-loss-threshold? uint32
        +--rw ccm-ttl?           uint8
        +--rw (flow-entropy)?
          | +--:(flow-entropy-null)
        +--rw MEP* [mep-id]
          | +--rw mep-id            MEP-id
          | +--rw mep-name?         string
          | +--rw mep-direction     MEP-direction
          | +--rw ccm-Tx-enable?    boolean
          | +--rw (mep-address)?
          |   | +--:(mac-address)
          |   |   | +--rw mac-address?      yang:mac-address
          |   |   +--:(ipv4-address)
          |   |   | +--rw ipv4-address?     inet:ipv4-address
          |   |   +--:(ipv6-address)
          |   |   | +--rw ipv6-address?     inet:ipv6-address
          | +--rw (context-id)?
          |   | +--:(context-null)
          |   |   +--rw context-null?      empty
          | +--rw Interface?       if:interface-ref
          | +--ro admin-status?    leafref
          | +--ro oper-status?     leafref
          | +--rw (flow-entropy)?
          |   | +--:(flow-entropy-null)
          | +--rw session* [user-cookie destination-mepid]
          |   | +--rw user-cookie          uint32
          |   | +--rw ttl?                 uint8
          |   | +--rw interval?            uint32
          |   | +--rw enable?              boolean
          |   | +--rw ecmp-choice?         ecmp-choices
          |   | +--rw destination-mepid    MEP-id

```

```

    +--rw destination-mep-address
    |   +--rw (mep-address)?
    |   |   +--:(mac-address)
    |   |   |   +--rw mac-address?      yang:mac-address
    |   |   +--:(ipv4-address)
    |   |   |   +--rw ipv4-address?      inet:ipv4-address
    |   |   +--:(ipv6-address)
    |   |   |   +--rw ipv6-address?      inet:ipv6-address
    +--ro ccm-rdi-indicator?              boolean
    +--ro ccm-xcon-count?                  oam-counter32
    +--ro ccm-xcon-Indicator?              boolean
    +--rw (context-id)?
    |   +--:(context-null)
    |   |   +--rw context-null?          empty
    +--rw (flow-entropy)?
    |   +--:(flow-entropy-null)
    +--rw outgoing-interface* [interface]
    |   +--rw interface      leafref
+--rw remote-MEP* [mep-id]
|   +--rw mep-id              uint32
|   +--rw (mep-address)?
|   |   +--:(mac-address)
|   |   |   +--rw mac-address?      yang:mac-address
|   |   +--:(ipv4-address)
|   |   |   +--rw ipv4-address?      inet:ipv4-address
|   |   +--:(ipv6-address)
|   |   |   +--rw ipv6-address?      inet:ipv6-address
|   +--rw mep-name?          string
|   +--rw ccm-rx-error-count? oam-counter32
+--rw MIP* [interface direction]
|   +--rw interface      if:interface-ref
|   +--rw direction      MEP-direction
+--ro ccm-rdi-indicator?    boolean
+--ro ccm-xcon-count?       oam-counter32
+--ro ccm-xcon-Indicator?   boolean
+--rw nested-oam-layer* [offset]
|   +--rw offset            int8
|   +--rw technology        identityref
|   +--rw md-name-format    MD-name-format
|   +--rw md-name?          binary
|   +--rw md-level          int32
|   +--rw ma-name-format    MA-name-format
|   +--rw ma-name           binary
rpcs:
  +---x ping
  |   +--ro input
  |   |   +--ro technology        identityref

```

```

+---ro md-name-format          MD-name-format
+---ro md-name?                binary
+---ro md-level                int32
+---ro ma-name-format          MA-name-format
+---ro ma-name                  binary
+---ro (context-id)?
|   +---:(context-null)
|       +---ro context-null?    empty
+---ro (flow-entropy)?
|   +---:(flow-entropy-null)
+---ro ttl?                     uint8
+---ro ecmp-choice?             ecmp-choices
+---ro sub-type?                identityref
+---ro outgoing-interfaces* [interface]
|   +---ro interface            if:interface-ref
+---ro source-mep
|   +---ro (mep-address)?
|       |   +---:(mac-address)
|       |       +---ro mac-address?    yang:mac-address
|       |       +---:(ipv4-address)
|       |           +---ro ipv4-address?    inet:ipv4-address
|       |           +---:(ipv6-address)
|       |               +---ro ipv6-address?    inet:ipv6-address
|       +---ro mep-id?                MEP-id
+---ro destination-mep
+---ro (mep-address)?
|   +---:(mac-address)
|       |   +---ro mac-address?    yang:mac-address
|       |       +---:(ipv4-address)
|       |           +---ro ipv4-address?    inet:ipv4-address
|       |           +---:(ipv6-address)
|       |               +---ro ipv6-address?    inet:ipv6-address
|       +---ro mep-id?                MEP-id
+---ro output
+---ro tx-packet-count?         oam-counter32
+---ro rx-packet-count?         oam-counter32
+---ro min-delay?               oam-counter32
+---ro average-delay?           oam-counter32
+---ro max-delay?               oam-counter32
+---x trace-route
+---ro input
+---ro technology                identityref
+---ro md-name-format          MD-name-format
+---ro md-name?                binary
+---ro md-level                int32
+---ro ma-name-format          MA-name-format
+---ro ma-name                  binary

```

```

+--ro (context-id)?
|   +---:(context-null)
|       +--ro context-null?          empty
+--ro (flow-entropy)?
|   +---:(flow-entropy-null)
+--ro ttl?                          uint8
+--ro command-sub-type?              identityref
+--ro ecmp-choice?                   ecmp-choices
+--ro outgoing-interfaces* [interface]
|   +--ro interface                  if:interface-ref
+--ro source-mep
|   +--ro (mep-address)?
|       |   +---:(mac-address)
|       |       |   +--ro mac-address?      yang:mac-address
|       |       +---:(ipv4-address)
|       |           |   +--ro ipv4-address?  inet:ipv4-address
|       |           +---:(ipv6-address)
|       |               |   +--ro ipv6-address?  inet:ipv6-address
|       +--ro mep-id?                MEP-id
+--ro destination-mep
|   +--ro (mep-address)?
|       |   +---:(mac-address)
|       |       |   +--ro mac-address?      yang:mac-address
|       |       +---:(ipv4-address)
|       |           |   +--ro ipv4-address?  inet:ipv4-address
|       |           +---:(ipv6-address)
|       |               |   +--ro ipv6-address?  inet:ipv6-address
|       +--ro mep-id?                MEP-id
+--ro output
|   +--ro response* [ttl]
|       +--ro ttl                    uint8
|       +--ro destination-mep
|           |   +--ro (mep-address)?
|           |       |   +---:(mac-address)
|           |       |       |   +--ro mac-address?      yang:mac-address
|           |       |       +---:(ipv4-address)
|           |           |   +--ro ipv4-address?  inet:ipv4-address
|           |           +---:(ipv6-address)
|           |               |   +--ro ipv6-address?  inet:ipv6-address
|           +--ro mep-id?                MEP-id
|       +--ro tx-packet-count?          oam-counter32
|       +--ro rx-packet-count?          oam-counter32
|       +--ro min-delay?                 oam-counter32
|       +--ro average-delay?             oam-counter32
|       +--ro max-delay?                 oam-counter32
notifications:
+---n CCM-RDI-notification

```

```
+++ro mep-id?          MEP-id
+++ro remote-mepid?    MEP-id
+++ro error-message?   string
```

Figure 5 data hierarchy of OAM

## 6. OAM YANG module

```
<CODE BEGINS> file "xxx.yang"

module gen-oam {
  namespace "urn:cisco:params:xml:ns:yang:gen-oam";
  prefix goam;

  import ietf-interfaces {
    prefix if;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }

  organization "IETF NETMOD (NETCONF Data Modeling ) Working Group";
  contact
    "Tissa Senevirathne tsenevir@cisco.com";
  description
    "This YANG module defines the generic configuration,
    statistics and rpc for OAM to be used within IETF in
    a protocol independent manner. Functional level
    abstraction is indendent with YANG modeling. It is
    assumed that each protocol maps corresponding
    abstracts to its native format.
    Each protocoal may extend the YANG model defined
    here to include protocol specific extensions";

  revision 2014-04-17 {
    description
      "Initial revision. - 02 version";
    reference "draft-tissa-netmod-oam";
  }

  identity technology-types {
    description
      "this is the base identity of technology types which are
      vpls, nvo3, TRILL, ipv4, ipv6, mpls";
  }

  identity ipv4 {
    base technology-types;
  }
}
```

```
    description
      "technology of ipv4";
  }

  identity ipv6 {
    base technology-types;
    description
      "technology of ipv6";
  }

  identity command-sub-type {
    description
      "defines different rpc command subtypes, e.g rfc792 ping
      vs udp ping, this is optional for most cases";
  }

  identity icmp-rfc792 {
    base command-sub-type;
    description
      "Defines the command subtypes for ICMP ping";
    reference "RFC 792";
  }

  typedef MEP-direction {
    type enumeration {
      enum "Up" {
        value 0;
      }
      enum "Down" {
        value 1;
      }
    }
  }

  typedef MEP-id {
    type uint32 {
      range "1..8191";
    }
    description
      "Defines type for MEPIDm range is 1..8191";
  }

  typedef CCM-Interval {
    default "interval-lmin";
    type enumeration {
      enum "interval-invalid" {
        value 0;
      }
    }
  }
```



```
    }
    enum "interval-300hz" {
        value 1;
    }
    enum "interval-10ms" {
        value 2;
    }
    enum "interval-100ms" {
        value 3;
    }
    enum "interval-1s" {
        value 4;
    }
    enum "interval-10s" {
        value 5;
    }
    enum "interval-1min" {
        value 6;
    }
    enum "interval-10min" {
        value 7;
    }
}
reference
    "802.2Q Rev5 or 802.ag, all of the above
    are standard enumeration from the 802.1Q";
description
    "IntervalInvalid - value 0
    Interval300Hz - Value 1
    Intervale10ms - value 2
    Intervall00ms - value3
    Intervall1s - value 4
    Intervall10s - value 5
    Intervall1min - value 6
    Intervall10min - value 7";
}

typedef ecmp-choices {
    type enumeration {
        enum "ecmp-use-platform-hash" {
            value 0;
        }
        enum "ecmp-use-round-robin" {
            value 1;
        }
    }
}
```

```
typedef MD-name-format {
  type enumeration {
    enum "ieee-reserved" {
      value 0;
    }
    enum "none" {
      value 1;
    }
    enum "dns-like-name" {
      value 2;
    }
    enum "mac-address-and-uint" {
      value 3;
      reference "802.1Q Rev5";
      description
        "Domain name 3 specifies domain name is mac-address + 2
octets.";
    }
  }
  reference "802.1Q";
  description
    "defines the domain name format";
}

typedef MA-name-format {
  type enumeration {
    enum "ieee-reserved" {
      value 0;
    }
    enum "primary-vid" {
      value 1;
    }
    enum "char-string" {
      value 2;
    }
    enum "unsigned-int16" {
      value 3;
    }
    enum "rfc2865-vpnid" {
      value 4;
    }
  }
  reference "802.1Q";
  description
    "Defines Format of MA-names";
}
```

```
typedef oam-counter32 {
  type yang:zero-based-counter32;
  description
    "defines 32 bit counter for OAM";
}

grouping mep-address {
  choice mep-address {
    case mac-address {
      leaf mac-address {
        type yang:mac-address;
      }
    }
    case ipv4-address {
      leaf ipv4-address {
        type inet:ipv4-address;
      }
    }
    case ipv6-address {
      leaf ipv6-address {
        type inet:ipv6-address;
      }
    }
  }
}

grouping maintenance-domain {
  status current;
  description
    "Defines the MA-domain group";
  reference "802.1Q Rev5";
  leaf technology {
    mandatory true;
    status current;
    type identityref {
      base technology-types;
    }
    description
      "Defines the technology";
  }
  leaf md-name-format {
    mandatory true;
    status current;
    description
      "Defines the maintenance domain name";
    type MD-name-format;
  }
}
```

```
        reference "802.1Q Rev5";
    }
    leaf md-name {
        status current;
        description
            "Defines the MA-Domain name. This is a binary (octet) string
            of 43 bytes";
        type binary {
            length "1..43";
        }
        reference "802.1Q Rev5";
    }
    leaf md-level {
        mandatory true;
        status current;
        description
            "Defines the MD-Level";
        type int32 {
            range "0..7";
        }
        reference "802.1Q Rev5 or 802.1ag";
    }
}

grouping ma-identifier {
    description
        "ma-identifier defines MAID parameters as defined in 8021Q";
    reference "IEEE 802.1Q Rev5";
    leaf ma-name-format {
        mandatory true;
        status current;
        description
            "This defines the MA name format 1 is no format,
            2 - dnslikename, 3- macaddress 4-CharString";
        type MA-name-format;
        reference "IEEE 802.1Q Rev 5";
    }
    leaf ma-name {
        mandatory true;
        description
            "Define the MA-Name according to the specified format.
            This is 43 byte string.";
        type binary {
            length "1..45";
        }
        reference "802.1Q Rve 5 or 8021ag Clause 21.6.5";
    }
}
```

```
    }

    grouping MEP {
        status current;
        description
            "Defines elements withing the MEP";
        reference "802.1Q Rev5";
        leaf mep-id {
            mandatory true;
            status current;
            description
                "Assign MEPID in the range of 1..8191";
            type MEP-id {
                range "1..8191";
            }
            reference "802.1Q Rev5";
        }
        leaf mep-name {
            type string;
            description
                "Defines textual name for MEP. This is not specified in IEEE
but
                defined in IETF OAM for ease of use";
        }
        leaf mep-direction {
            type MEP-direction;
            mandatory true;
        }
        leaf ccm-Tx-enable {
            type boolean;
            default "false";
        }
        uses mep-address;
        uses context-id;
        leaf Interface {
            type if:interface-ref;
            description
                "Interface name as defined by ietf-interfaces";
        }
    }

    grouping CCM-defect-stats {
        description
            "Contains all of the CCM related defect stats";
        leaf ccm-rdi-indicator {
            config false;
            type boolean;
        }
    }
}
```

```
    description
      "True indicate one or more of the MEP have seen RDI
       flag set from remote MEP";
  }
  leaf ccm-xcon-count {
    config false;
    type oam-counter32;
    description
      "Number of times cross connect errors are seen";
  }
  leaf ccm-xcon-Indicator {
    config false;
    type boolean;
    description
      "There is currently cross connect error seen since last
       clearing of the variable";
  }
}

grouping monitor-stats {
  leaf tx-packet-count {
    type oam-counter32;
    description
      "Transmitted Packet count";
  }
  leaf rx-packet-count {
    type oam-counter32;
    description
      "Received packet count";
  }
  leaf min-delay {
    units "milliseconds";
    type oam-counter32;
    description
      "Delay is specified in milliseconds";
  }
  leaf average-delay {
    units "milliseconds";
    type oam-counter32;
    description
      "average delay in milliseconds";
  }
  leaf max-delay {
    type oam-counter32;
    units "millisecond";
  }
}
```

```
grouping MIP {
  description
    "defines MIP";
  leaf interface {
    type if:interface-ref;
  }
  leaf direction {
    type MEP-direction;
  }
}

grouping nested-oam-layer {
  leaf offset {
    type int8 {
      range "1..7";
    }
    description
      "defines nested OAM layer offset
      +1 is the layer immediatly above
      -1 is the layer immediatly below";
  }
  uses maintenance-domain;
  uses ma-identifier;
}

grouping interface-status {
  description
    "collection of interface related status";
  leaf admin-status {
    config false;
    type leafref {
      path "/if:interfaces-state/if:interface/if:admin-status";
    }
    description
      "oper status from ietf-interface module";
  }
  leaf oper-status {
    config false;
    type leafref {
      path "/if:interfaces-state/if:interface/if:oper-status";
    }
    description
      "oper status from ietf-interface module";
  }
}
```

```
grouping context-id {
  description
    "grouping for context id, this will be augmented
    by others who use this component";
  choice context-id {
    default "context-null";
    case context-null {
      description
        "this is a place holder when no context is needed";
      leaf context-null {
        type empty;
        description
          "there is no context define";
      }
    }
  }
}

grouping flow-entropy {
  description
    "defines the grouping statement for flow-entropy";
  choice flow-entropy {
    case flow-entropy-null;
  }
}

container domains {
  status current;
  config true;
  description
    "Contains configuration related data. Within the container
    is list of fault domains. Wihin each domian has List of MA.";
  list domain {
    uses maintenance-domain {
      status current;
    }
  }
  key "md-name technology";
  ordered-by system;
  status current;
  config true;
  description
    "Define the list of Domains within the IETF-OAM";
  container MAS {
    presence
      "Indicates creation of MA within the Domain
      There can be more than one MA within a specified domain";
    status current;
  }
}
```



```
config true;
description
  "This container defines MA, within that have multiple MA
   and within MA have MEP, MIP";
list MA {
  ordered-by system;
  status current;
  config true;
  key "ma-name";
  uses ma-identifier;
  uses context-id;
  leaf ccm-Interval {
    default "interval-invalid";
    description
      "Defines CCM Interval 0- Means disable
       1 - CCM are sent 3 1/3 ms
       2 - CCM are sent every 10 ms
       3- CCM are sent every 100 ms
       4- CCM are sent every 1 s
       5 - CCM are sent every 10 s
       6 - CCM are sent every 1 minute
       7- CCM are sent every 10 mins";
    type CCM-Interval;
    reference "802.1Q Rev5 and 802.1ag";
  }
  leaf ccm-loss-threshold {
    default "3";
    type uint32;
    description
      "number of consecutive CCM messages missed before
       declaring RDI fault. This is monitored per each
       remote MEP";
  }
  leaf ccm-ttl {
    type uint8;
    default "255";
  }
  uses flow-entropy;
  list MEP {
    key "mep-id";
    ordered-by system;
    status current;
    config true;
    description
      "contain list of MEPS";
    uses MEP {
      status current;
    }
  }
}
```

```
}
uses interface-status {
  description
    "status of associated interface";
}
uses flow-entropy;
list session {
  key "user-cookie destination-mepid";
  ordered-by user;
  config true;
  description
    "per session basis create the monitoring";
  leaf user-cookie {
    config true;
    type uint32;
    description
      "user need to specify some cookie to identify
       multiple sessions between two MEPs";
  }
  leaf ttl {
    config true;
    type uint8;
    default "255";
  }
  leaf interval {
    units "milliseconds";
    default "1000";
    type uint32;
    description
      "In milli seconds. 0 means continous";
  }
  leaf enable {
    default "false";
    config true;
    type boolean;
    description
      "enable or disable a monitor session";
  }
  leaf ecmp-choice {
    config true;
    type ecmp-choices;
    description
      "0 means use the specified interface
       1 means use round robin";
  }
  leaf destination-mepid {
    type MEP-id;
  }
}
```

```
    }
    container destination-mep-address {
      uses mep-address;
    }
    uses CCM-defect-stats;
    uses context-id;
    uses flow-entropy;
    list outgoing-interface {
      config true;
      key "interface";
      leaf interface {
        type leafref {
          path "/if:interfaces/if:interface/if:name";
        }
      }
      config true;
    }
  }
}
list remote-MEP {
  key "mep-id";
  ordered-by system;
  status current;
  config true;
  description
    "list all of the remote MEP within the MA";
  leaf mep-id {
    mandatory true;
    status current;
    description
      "Assign MEPID in the range of 1..8191";
    config true;
    type uint32;
    reference "802.1Q Rev5";
  }
  uses mep-address;
  leaf mep-name {
    type string;
    description
      "Defines textual name for MEP. This is not
       specified in IEEE but defined in IETF OAM
       for ease of use";
  }
  leaf ccm-rx-error-count {
    type oam-counter32;
    description
      "counts number of CCM packets that was
```

```

        expected but not received";
    }
}
list MIP {
    key "interface direction";
    uses MIP;
}
uses CCM-defect-stats {
    description
        "CCM defect stats capture at MA level
        This will contain aggregate stats from all MEP";
}
list nested-oam-layer {
    key "offset";
    description
        "List of OAM layers above and below that are related to
        current MA. This allow users to easily navigate up and
down
        to effeciently troubleshoot a connectivity issue";
    uses nested-oam-layer;
}
}
}
}
}
notification CCM-RDI-notification {
    description
        "When RDI is received this notificiation is sent";
    leaf mep-id {
        type MEP-id;
        description
            "Indicate which MEP is seeing the error";
    }
    leaf remote-mepid {
        type MEP-id;
        description
            "Who is seeing the error (if known) if unknown make it 0.";
    }
    leaf error-message {
        type string {
            length "0..255";
        }
        description
            "Error message to indicate more details.";
    }
}
}
rpc ping {

```

```
description
  "Generates Ping and return response";
input {
  uses maintenance-domain {
    description
      "Specifies the MA-domain";
  }
  uses ma-identifier {
    description
      "identifies the Maintenance association";
  }
  uses context-id;
  uses flow-entropy;
  leaf ttl {
    type uint8;
    default "255";
  }
  leaf ecmp-choice {
    type ecmp-choices;
    description
      "0 means use the specified interface
       1 means use round robin";
  }
  leaf sub-type {
    type identityref {
      base command-sub-type;
    }
    description
      "defines different command types";
  }
  list outgoing-interfaces {
    key "interface";
    leaf interface {
      type if:interface-ref;
    }
  }
  container source-mep {
    uses mep-address;
    leaf mep-id {
      type MEP-id;
    }
  }
  container destination-mep {
    uses mep-address;
    leaf mep-id {
      type MEP-id;
    }
  }
}
```

```
    }
  }
  output {
    uses monitor-stats {
      description
        "Stats of Ping is same as that of monitor sessions";
    }
  }
}
rpc trace-route {
  description
    "Generates Trace-route and return response. Starts with TTL
    of one and increment by one at each hop. Untill destination
    reached or TTL reach max valune";
  input {
    uses maintenance-domain {
      description
        "Specifies the MA-domain";
    }
    uses ma-identifier {
      description
        "identifies the Maintenance association";
    }
    uses context-id;
    uses flow-entropy;
    leaf ttl {
      type uint8;
      default "255";
    }
    leaf command-sub-type {
      type identityref {
        base command-sub-type;
      }
      description
        "defines different command types";
    }
    leaf ecmp-choice {
      type ecmp-choices;
      description
        "0 means use the specified interface
        1 means use round robin";
    }
    list outgoing-interfaces {
      key "interface";
      leaf interface {
        type if:interface-ref;
      }
    }
  }
}
```

```
    }
    container source-mep {
      uses mep-address;
      leaf mep-id {
        type MEP-id;
      }
    }
    container destination-mep {
      uses mep-address;
      leaf mep-id {
        type MEP-id;
      }
    }
  }
  output {
    list response {
      key "ttl";
      leaf ttl {
        type uint8;
      }
      container destination-mep {
        uses mep-address;
        leaf mep-id {
          type MEP-id;
        }
      }
      uses monitor-stats;
    }
  }
}
```

<CODE ENDS>

Figure 6 YANG module of OAM

## 7. Base Mode for IP

The Base Mode defines default configuration that MUST be present in the devices that comply with this document. Base Mode allows users to have "zero-touch" experience. Several parameters require technology specific definition.

### 7.1. MEP Address

In the Base Mode of operation, the MEP Address is the IP address of the interface on which the MEP is located.

## 7.2. MEP ID for Base Mode

In the Base Mode of operation, each device creates a single UP MEP associated with a virtual OAM port with no physical layer (NULL PHY). The MEPID associated with this MEP is zero (0). The choice of MEP-ID zero is explained below.

MEPID is 2 octet field. It is never used on the wire except when using CCM. Ping, traceroute and session monitoring does not use the MEPID on its message header. It is important to have method that can derive MEP ID of base mode in an automatic manner with no user intervention. IP address cannot be directly used for this purpose as the MEP ID is much smaller field. For Base Mode of IP we propose to use MEP ID zero (0) as the default MEP-ID.

CCM packet use MEP-ID on the payload. CCM MUST NOT be used in the Base Mode for IP. Hence CCM MUST be disabled on the Maintenance Association of the Base Mode.

If CCM is required, users MUST configure a separate Maintenance association and assign unique value for the corresponding MEP IDs.

[8021Q] CFM defines MEP ID as an unsigned integer in the range 1 to 8191. In this document we propose to extend the range to 0 to 65535. Value 0 is reserved for MEP ID of Base Mode of IP and MUST NOT be used for other purposes.

## 7.3. Maintenance Domain

Default MD-LEVEL is set to 3.

## 7.4. Maintenance Association

MAID [8021Q] has a flexible format and includes two parts: Maintenance Domain Name and Short MA name. In the Based Mode of operation, the value of the Maintenance Domain Name must be the character string "GenericBaseMode" (excluding the quotes "). In Base Mode operation Short MA Name format is set to 2-octet integer format (value 3 in Short MA Format field [8021Q]) and Short MA name set to 65532 (0xFFFC).

## 8. Security Considerations

TBD



## 9. IANA Considerations

This document registers the following namespace URI in the IETF XML registry.

URI:TBD

## 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.
- [RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.
- [8021Q] IEEE, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks", IEEE Std 802.1Q-2011, August, 2011.

### 10.2. Informative References

- [Y1731] ITU, "OAM functions and mechanisms for Ethernet based networks", ITU-T G.8013/Y.1731, July, 2011.
- [RFC7174] Salam, S., et.al., "TRILL OAM Framework", RFC7174, May 2014.
- [RFC6291] Andersson, L., et.al., "Guidelines for the use of the "OAM" Acronym in the IETF" RFC 6291, June 2011.
- [RFC6325] Perlman, R., et.al., "Routing Bridges (Rbridges): Base Protocol Specification", RFC 6325, July 2011.
- [OAMOVW] Mizrahi, T., et.al., "An Overview of Operations, Administration, and Maintenance (OAM) Tools", draft-ietf-opsawg-oam-overview-16, Work in Progress, March 2014.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

- [RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", RFC 4379, February 2006.
- [RFC6371] Busi, I., et.al., "Operations, Administration, and Maintenance Framework for MPLS-Based Transport Networks", RFC 6317, September 2011.
- [TRILLOAMFM] Senevirathne, T., et.al., "TRILL OAM Fault Management", draft-ietf-trill-oam-fm, Work in Progress, May 2014.

## 11. Acknowledgments

Giles Heron came up with the idea of developing a YANG model as a way of creating a unified OAM API set (interface), work in this document is largely an inspiration of that. Alexander Clemm provided many valuable tips, comments and remarks that helped to refine the YANG model presented in this document.

This document was prepared using 2-Word-v2.0.template.dot.

## Authors' Addresses

Tissa Senevirathne  
CISCO Systems  
375 East Tasman Drive.  
San Jose, CA 95134  
USA.

Phone: 408-853-2291  
Email: tsenevir@cisco.com

Norman Finn  
CISCO Systems  
510 McCarthy Blvd  
Milpitas, CA 95035.

Email: nfinn@cisco.com

Deepak Kumar  
CISCO Systems  
510 McCarthy Blvd  
Milpitas, CA 95035.

Email: dekumar@cisco.com

Samer Salam  
CISCO Systems  
595 Burrard St. Suite 2123  
Vancouver, BC V7X 1J1, Canada

Email: ssalam@cisco.com

Carlos Pignataro  
Cisco Systems  
7200-12 Kit Creek Rd,  
Research Triangle Park, NC 27709

Email: cpignata@cisco.com



NETMOD WG  
Internet-Draft  
Intended status: Informational  
Expires: January 03, 2015

Clyde Wildes  
Cisco Systems  
Agrahara Kiran Koushik  
Brocade Communication Systems  
July 03, 2014

SYSLOG YANG model  
draft-wildes-netmod-syslog-model-01

Abstract

This document describes a data model for Syslog protocol which is used to convey event notification messages.

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 January 03, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Definitions and Acronyms . . . . .	3
2. Problem Statement . . . . .	3
3. Design of the SYSLOG Model . . . . .	3
3.1. SYSLOG Module . . . . .	4
4. SYSLOG YANG Models . . . . .	6
4.1. SYSLOG TYPES Module . . . . .	6
4.2. SYSLOG module . . . . .	10
4.3. A SYSLOG Example . . . . .	16
5. Implementation Status . . . . .	17
6. Security Considerations . . . . .	17
7. IANA Considerations . . . . .	18
8. Acknowledgements . . . . .	18
9. Change log [RFC Editor: Please remove] . . . . .	18
10. References . . . . .	18
Authors' Addresses . . . . .	19

## 1. Introduction

Operating systems, processes and applications generate messages indicating their own status or the occurrence of events. These messages are useful for managing and/or debugging the network and its services. The BSD Syslog protocol is a widely adopted protocol that is used for transmission and processing of the messages.

Since each process, application and operating system was written somewhat independently, there is little uniformity to the content of Syslog messages. For this reason, no assumption is made upon the formatting or contents of the messages. The protocol is simply designed to transport these event messages. No acknowledgement of the receipt is made.

Essentially, a Syslog process receives messages (from the kernel, processes, applications or other Syslog processes) and processes those. The processing involves logging to a local file, displaying on console, user terminal, and/or relaying to syslog processes on other machines. The processing is determined by the "facility" that originated the message and the "severity" assigned to the message by the facility.

We are using definitions of Syslog protocol from [RFC3164] in this draft.

### 1.1. Definitions and Acronyms

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

UDP: User Datagram Protocol

VRF: Virtual Routing and Forwarding

## 2. Problem Statement

This document defines a YANG [RFC6020] configuration data model that may be used to monitor and control one or more syslog processes running on a system. YANG models can be used with network management agents such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

This module makes use of the YANG "feature" construct which allows implementations to support only those Syslog features that lie within their capabilities.

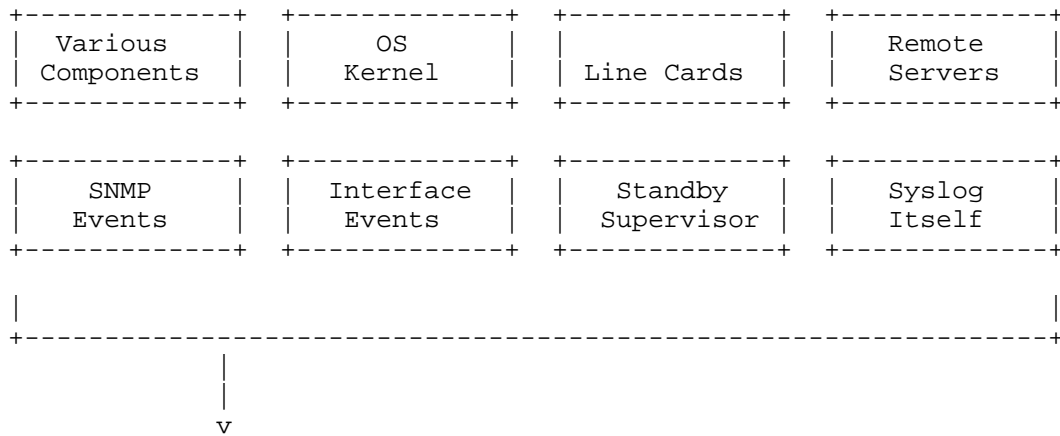
## 3. Design of the SYSLOG Model

The syslog model was designed by comparing various syslog features implemented by various vendors' in different implementations.

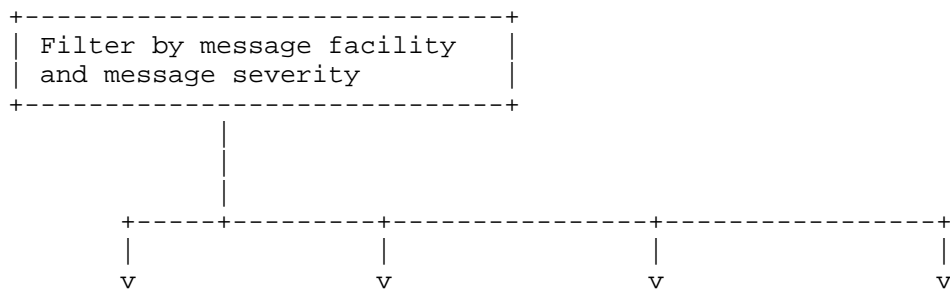
This draft addresses the common leafs between all vendors and creates a common model, which can be augmented with proprietary features, if necessary. The base model is designed to be very simple for maximum flexibility.

Syslog consists of message producers, a group level suppression filter, and message distributors. The following diagram shows syslog messages flowing from a message producer, through the group level suppression filter, and if passed by the group filter to message distributors where further suppression filtering can take place.

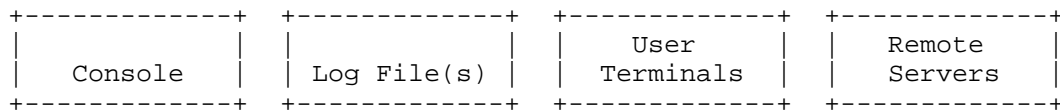
## Message Producers



## Group Level Suppression



## Message Distributors



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
      | +--rw logging-severities [facility]
      |   +--rw facility      identityref
      |   +--rw severity?    syslogtypes:Severity
    +--rw console-logging
      | +--rw (logging-level-scope)?
      |   +--:(all-facilities)
      |   | +--rw logging-severity?    syslogtypes:Severity
      |   +--:(facility)
      |     +--rw logging-severities [facility]
      |       +--rw facility      identityref
      |       +--rw severity?    syslogtypes:Severity
    +--rw file-logging
      | +--rw file-name          string
      | +--rw file-size?        uint32
      | +--rw (logging-scope)?
      |   +--:(all-facilities)
      |   | +--rw logging-severity?    syslogtypes:Severity
      |   +--:(facility)
      |     +--rw logging-severities [facility]
      |       +--rw facility      identityref
      |       +--rw severity?    syslogtypes:Severity
    +--rw remote-logging
      | +--rw remote-logging-destination [destination]
      |   +--rw destination          string
      |   +--rw logging-severities [facility]
      |     +--rw facility      identityref
      |     +--rw severity?    syslogtypes:Severity
      | +--rw source-interface?    string
      | +--rw vrf-name?            string
    +--rw terminal-logging
      | +--rw (user-scope)?
      |   +--:(all-users)
      |   | +--rw all-users
      |   |   +--rw (logging-scope)?
      |   |     +--:(all-facilities)
      |   |     | +--rw logging-severity?    syslogtypes:Severity
      |   |     +--:(facility)
      |   |       +--rw logging-severities [facility]
      |   |       +--rw facility      identityref
      |   |       +--rw severity?    syslogtypes:Severity
      |   +--:(per-user)
      |     +--rw user-name [uname]
      |       +--rw uname          string
      |       +--rw (logging-scope)?
      |         +--:(all-facilities)
      |         | +--rw logging-severity?    syslogtypes:Severity
      |         +--:(facility)
      |           +--rw logging-severities [facility]
      |             +--rw facility      identityref
      |             +--rw severity?    syslogtypes:Severity

```

## 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 2014-06-03 {
    description
      "Version 1.0";
  }

  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;
}
```

```
identity user {
    base syslog-facility;
}

identity mail {
    base syslog-facility;
}

identity daemon {
    base syslog-facility;
}

identity auth {
    base syslog-facility;
}

identity syslog {
    base syslog-facility;
}

identity lpr {
    base syslog-facility;
}

identity news {
    base syslog-facility;
}

identity uucp {
    base syslog-facility;
}

identity cron {
    base syslog-facility;
}

identity authpriv {
    base syslog-facility;
}

identity ftp {
    base syslog-facility;
}

identity ntp {
    base syslog-facility;
}

identity audit {
    base syslog-facility;
}
```

```
    identity console {
        base syslog-facility;
    }

    identity cron2 {
        base syslog-facility;
    }

    identity local0 {
        base syslog-facility;
    }

    identity local1 {
        base syslog-facility;
    }

    identity local2 {
        base syslog-facility;
    }

    identity local3 {
        base syslog-facility;
    }

    identity local4 {
        base syslog-facility;
    }

    identity local5 {
        base syslog-facility;
    }

    identity local6 {
        base syslog-facility;
    }

    identity local7 {
        base syslog-facility;
    }
}
```

## 4.2. SYSLOG module

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

  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 2014-06-10 {
    description
      "Initial revision.";
  }

  feature global-logging {
    description
      "This feature represents the ability to adjust
      log message severity per logging facility on the global level.";
  }

  feature console-facility-logging-config {
    description
      "This feature represents the ability to adjust
      log message severity per logging facility for console logging.";
  }

  feature file-logging {
    description
      "This feature represents the ability to log
      messages into a file.";
  }
}
```

```
feature file-facility-logging-config {
  description
    "This feature represents the ability to adjust
    log message severity per logging facility for file logging.";
}

feature terminal-facility-logging-config {
  description
    "This feature represents the ability to adjust
    log message severity per logging facility for terminal logging.";
}

feature terminal-facility-user-logging-config {
  description
    "This feature represents the ability to adjust
    log message settings for individual terminal users.";
}

feature use-vrf {
  description
    "This feature allows logging of messages to a particular VRF.";
}

grouping facility-logging {
  description
    "This grouping defines a list of facility-severity pairs.  Messages
    from a facility in the list that have the corresponding specified
    severity level or higher will be logged.";
  list logging-severities {
    description
      "This list describes a collection of Syslog facilities.";
    key "facility";
    leaf facility {
      type identityref {
        base syslogtypes:syslog-facility;
      }
      description
        "The leaf uniqueely identifies a Syslog facility.";
    }
    leaf severity {
      type syslogtypes:Severity;
      description
        "This leaf specifies the severity of Syslog messages
        for this facility.";
    }
  }
}
```

```
container syslog {
  description
    "This container describes the configuration parameters for Syslog.";
  config true;
  container global-logging {
    if-feature global-logging;
    description
      "This container describes the configuration parameters for global
      logging.";
    uses facility-logging;
  }
  container console-logging {
    description
      "This container describes the configuration parameters for console
      logging.";
    choice logging-level-scope {
      description
        "This choice describes the option to specify all facilities or
        a specific facility.";
      case all-facilities {
        description
          "This case specifies all facilities.";
        leaf logging-severity {
          type syslogtypes:Severity;
          description
            "This leaf specifies the severity of Syslog messages
            for all facilities.";
        }
      }
      case facility {
        description
          "This case specifies a specific facility.";
        if-feature console-facility-logging-config;
        uses facility-logging;
      }
    }
  }
}
```



```
container file-logging {
  if-feature file-logging;
  description
    "This container describes the configuration parameters for file
    logging configuration.";
  leaf file-name {
    mandatory true;
    type string;
    description
      "This leaf specifies the name of the log file.";
  }
  leaf file-size {
    type uint32;
    description
      "This leaf specifies the log file size.";
  }
  choice logging-scope {
    description
      "This choice describes the option to specify all facilities or
      a specific facility.";
    case all-facilities {
      description
        "This case specifies all facilities.";
      leaf logging-severity {
        type syslogtypes:Severity;
        description
          "This leaf specifies the severity of Syslog messages
          for all facilities.";
      }
    }
    case facility {
      description
        "This case specifies a specific facility.";
      if-feature file-facility-logging-config;
      uses facility-logging;
    }
  }
}

container remote-logging {
  description
    "This container describes the configuration parameters for the remote
    logging configuration.";
  list remote-logging-destination {
    description
      "This list describes a collection of remote logging destinations.";
    key "destination";
    leaf destination {
      type string;
      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 facility-logging;
    leaf source-interface {
        description
            "This leaf sets the source interface for the remote Syslog server.
            Either the interface name or the interface IP address can be
            specified.";
        type string;
    }
    leaf vrf-name {
        if-feature use-vrf;
        type string;
        description
            "This leaf specifies the name of the virtual routing facility
            (VRF).";
    }
}
}
container terminal-logging {
    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.";
        case all-users {
            description
                "This case specifies all users.";
            container all-users {
                choice logging-scope {
                    description
                        "This choice describes the option to specify all facilities or
                        a specific facility.";
                    case all-facilities {
                        description
                            "This case specifies all facilities.";
                        leaf logging-severity {
                            type syslogtypes:Severity;
                            description
                                "This leaf specifies the severity of Syslog messages
                                for all facilities.";
                        }
                    }
                }
            }
        }
    }
}
```

```

        case facility {
            description
                "This case specifies a specific facility.";
            if-feature terminal-facility-logging-config;
            uses facility-logging;
        }
    }
}
case per-user {
    description
        "This case specifies a specific user.";
    if-feature terminal-facility-user-logging-config;
    list user-name {
        description
            "This list describes a collection of user names.";
        key "uname";
        leaf uname {
            type string;
            description
                "This leaf uniquely describes a user name.";
        }
        choice logging-scope {
            description
                "This choice describes the option to specify all facilities or
                 a specific facility.";
            case all-facilities {
                description
                    "This case specifies all facilities.";
                leaf logging-severity {
                    type syslogtypes:Severity;
                    description
                        "This leaf specifies the severity of Syslog messages
                         for all facilities.";
                }
            }
            case facility {
                description
                    "This case specifies a specific facility.";
                if-feature terminal-facility-logging-config;
                uses facility-logging;
            }
        }
    }
}
}
}
}
}
}
}

```

#### 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:cisco:params:xml:ns:yang:syslog">
        <global-logging>
          <facility>syslogtypes:kern</facility>
          <severity>syslogtypes:critical</severity>
          <facility>syslogtypes:auth</facility>
          <severity>syslogtypes:error</severity>
        </global-logging>
        <console-logging>
          <logging-severity>syslogtypes:critical</logging-severity>
        </console-logging>
      </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	logging level cron 2
syslog:console-logging	logging console 1
syslog:file-logging	logging logfile mylog.log 2 4096
syslog:terminal-logging	logging monitor 2
syslog:remote-logging	*logging server server.cisco.com 2 facility user use-vrf management *logging source-interface loopback 0
cisco-syslog:logging-timestamp-config	logging timestamp milli-seconds
cisco-syslog:origin-id-cfg	logging origin-id string abcdef
cisco-syslog:module-logging	logging module 1

\*The "logging server" and "logging source-interface" commands were combined into one base model leaf.

The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

## 6. Security Considerations

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

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

TBD: List specific Subtrees and data nodes and their sensitivity/vulnerability.

## 7. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:syslog

Registrant Contact: The IESG.

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

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

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

## 8. Acknowledgements

## 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, 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.

Authors' Addresses

Clyde Wildes  
Cisco Systems Inc.

Email: cwildes@cisco.com

Kiran Agrahara Sreenivasa  
Brocade Communications Systems

Email: kkoushik@brocade.com

NETMOD Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 5, 2014

Y. Yang  
W. Dec  
Cisco Systems  
July 4, 2014

A YANG Data Model for Location Information  
<draft-yang-netmod-location-00.txt>

Abstract

This document defines a YANG data model for RFC 5139 civic location information.

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 December 5, 2014.



Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	3
1.2. Tree Diagrams . . . . .	3
2. Data Model . . . . .	4
2.1. Civic Address . . . . .	4
2.2. Geospatial Coordinates . . . . .	4
2.3. Location Data Grouping . . . . .	5
3. YANG Modules . . . . .	6
3.1. IANA Maintained Civic Address Type YANG Submodule . . . . .	6
3. IANA(??) Maintained Geo URI YANG Submodule . . . . .	12
4. Location YANG module . . . . .	13
5. Security Considerations . . . . .	19
6. IANA Considerations . . . . .	19
6.1. URI Registrations . . . . .	20
6.2. YANG Module Registrations . . . . .	20
7. Normative References . . . . .	20
Appendix A. Acknowledgments . . . . .	21
Authors' Addresses . . . . .	22

## 1. Introduction

The physical location of objects, such as network devices, persons, events and etc, can be described using location information defined in[GEOPRIV]. This document defines a YANG [YANG] data model for such location information.

## 1.1. 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 [KEYWORD].

## 1.2. 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:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!"

means a presence container, and "\*" denotes a list and leaf-list.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2. Data Model

The yang location data model consists of the definition of the data-types for civic addresses, and geospatial coordinates, and a grouping of these types. A simple, non identity based, data-type definition is used to simplify augmentations as well as the instantiated data structure and its processing. A grouping of the data-types is provided and intended to be used by importing modules intending to express location information about a particular subject.

### 2.1. Civic Address

A civic address includes various types of information about the country, administrative units such as states, provinces, and cities, as well as street addresses, postal community names, building information, and etc [CIVIC].

These civic address types are registered and maintained in IANA's "Civic Address Types Registry" [CATYPE-REGISTRY], and the corresponding YANG data types are defined in the iana-civic-address-type YANG submodule.

Whenever a new civic address type is added to the "Civic Address Types Registry", the iana-civic-address-type submodule should be updated by IANA.

### 2.2. Geospatial Coordinates

Geospatial coordinates indicate longitude, latitude, altitude, and etc.

[GEO-URI] specifies a Uniform Resource Identifier (URI) for geographic locations using the 'geo' scheme name, in a compact, simple, human-readable, and protocol-independent way. The default coordinate reference system used is the World Geodetic System 1984 (WGS-84). The "'geo' URI Parameters" registry [GEO-REGISTRY] is maintained by IANA for the <parameter> component of the 'geo' URI, and the sub-registry "'geo' URI 'crs' Parameter Values" [CRS-REGISTRY] is maintained by IANA for the 'crs' parameter.

The iana-geo-uri-type submodule defines a data type that serializes

the "geo" URI into YANG format.

Whenever a new "geo" URI parameter is added to the [GEO-REGISTRY], or a new CRS parameter value is added to the [CRS-REGISTRY], The iana-geo-uri-type YANG module should be updated by IANA.

### 2.3. Location Data Grouping

A containerized grouping of the location data is provided as per the following structure:

```
module: ietf-location
+--rw location
  +--rw geo-uri?    geo-uri
  +--rw country?    country
  +--rw CA0?        CA0
  +--rw A1?         A1
  +--rw A2?         A2
  +--rw A3?         A3
  +--rw A4?         A4
  +--rw A5?         A5
  +--rw A6?         A6
  +--rw PRD?        PRD
  +--rw POD?        POD
  +--rw STS?        STS
  +--rw HNO?        HNO
  +--rw HNS?        HNS
  +--rw LMK?        LMK
  +--rw LOC?        LOC
  +--rw NAM?        NAM
  +--rw PC?         PC
  +--rw BLD?        BLD
  +--rw UNIT?       UNIT
  +--rw FLR?        FLR
  +--rw ROOM?       ROOM
  +--rw PLC?        PLC
  +--rw PCN?        PCN
  +--rw POBOX?      POBOX
  +--rw ADDCODE?    ADDCODE
  +--rw SEAT?       SEAT
  +--rw RD?         RD
  +--rw RDSEC?      RDSEC
  +--rw RDBR?       RDBR
  +--rw RDSUBBR?    RDSUBBR
  +--rw PRM?        PRM
  +--rw POM?        POM
  +--rw PN?         PN
```

+++rw MP?	MP
+++rw STP?	STP
+++rw HNP?	HNP
+++rw STPS?	STPS
+++rw LMKP?	LMKP
+++rw CA128?	CA128

### 3. YANG Modules

#### 3.1. IANA Maintained Civic Address Type YANG Submodule

```
<CODE BEGINS> file "iana-civic-address-type@2014-05-08.yang"

submodule iana-civic-address-type {
  belongs-to "ietf-location" {
    prefix loc;
  }

  organization "IANA";
  contact
    "      Internet Assigned Numbers Authority

    Postal: ICANN
           4676 Admiralty Way, Suite 330
           Marina del Rey, CA 90292

    Tel:    +1 310 823 9358
    E-Mail: iana@iana.org";
  description
    "This YANG module defines the iana-civic-address-type typedef,
    which contains YANG definitions for IANA-registered civic
    address types.

    This YANG module is maintained by IANA, and reflects the
    'Civic Address Types (CATypes)' registry.

    The latest revision of this YANG module can be obtained from
    the IANA web site.

    Requests for new values should be made to IANA via
    email (iana@iana.org).

    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).";

revision 2014-05-08 {
  description
    "Initial revision.";
}

typedef iana-civic-address-type {
  type string;
  description
    "A civic address type as defined by the IANA Civic Address
    Types registry.";
}

typedef CA0 {
  type iana-civic-address-type;
  description
    "Language.";
}

typedef A1 {
  type iana-civic-address-type;
  description
    "National subdivisions:
    state, canton, region, province, prefecture";
}

typedef A2 {
  type iana-civic-address-type;
  description
    "county, parish, gun (JP), district (IN)";
}

typedef A3 {
  type iana-civic-address-type;
  description
    "city, township, shi (JP)";
}

typedef A4 {
  type iana-civic-address-type;
  description
    "city division, borough, city district, ward, chou (JP)";
}
```

```
typedef A5 {
    type iana-civic-address-type;
    description
        "neighborhood, block";
}

typedef A6 {
    type iana-civic-address-type;
    description
        "group of streets below the neighborhood level";
}

typedef PRD {
    type iana-civic-address-type;
    description
        "leading street direction";
}

typedef POD {
    type iana-civic-address-type;
    description
        "trailing street suffix";
}

typedef STS {
    type iana-civic-address-type;
    description
        "street suffix or type";
}

typedef HNO {
    type iana-civic-address-type;
    description
        "house number";
}

typedef HNS {
    type iana-civic-address-type;
    description
        "house number suffix";
}

typedef LMK {
    type iana-civic-address-type;
    description
        "landmark or vanity address";
}
```

```
typedef LOC {
    type iana-civic-address-type;
    description
        "additional location information";
}

typedef NAM {
    type iana-civic-address-type;
    description
        "name (residence and office occupant)";
}

typedef PC {
    type iana-civic-address-type;
    description
        "postal/zip code";
}

typedef BLD {
    type iana-civic-address-type;
    description
        "building (structure)";
}

typedef UNIT {
    type iana-civic-address-type;
    description
        "unit (apartment, suite)";
}

typedef FLR {
    type iana-civic-address-type;
    description
        "floor";
}

typedef ROOM {
    type iana-civic-address-type;
    description
        "room number";
}

typedef PLC {
    type iana-civic-address-type;
    description
        "type of place";
}
```



```
typedef PCN {
    type iana-civic-address-type;
    description
        "postal community name";
}

typedef POBOX {
    type iana-civic-address-type;
    description
        "post office box (P.O. Box)";
}

typedef ADDCODE {
    type iana-civic-address-type;
    description
        "additional code";
}

typedef SEAT {
    type iana-civic-address-type;
    description
        "seat (desk, cubicle, workstation)";
}

typedef RD {
    type iana-civic-address-type;
    description
        "Primary road or street";
}

typedef RDSEC {
    type iana-civic-address-type;
    description
        "Road section";
}

typedef RDBR {
    type iana-civic-address-type;
    description
        "Road branch";
}

typedef RDSUBBR {
    type iana-civic-address-type;
    description
        "Road sub-branch";
}
```

```
typedef PRM {
    type iana-civic-address-type;
    description
        "Street name pre-modifier";
}

typedef POM {
    type iana-civic-address-type;
    description
        "Street name post-modifier";
}

typedef PN {
    type iana-civic-address-type;
    description
        "Post number that is attributed to a lamp post or utility
        pole.";
}

typedef MP {
    type iana-civic-address-type;
    description
        "Milepost: a marker indicating distance to or from a place
        (often a town).";
}

typedef STP {
    type iana-civic-address-type;
    description
        "Street Type Prefix.";
}

typedef HNP {
    type iana-civic-address-type;
    description
        "House Number Prefix.";
}

typedef STPS {
    type iana-civic-address-type;
    description
        "Separator between the existing Street Type Prefix (STP) and
        the Road Name (RD) elements.";
}

typedef LMKP {
    type iana-civic-address-type;
    description
```

```
        "Component part of a Complete Landmark Name.";
    }

    typedef CA128 {
        type iana-civic-address-type;
        description
            "script";
    }
}
```

### 3. IANA(??) Maintained Geo URI YANG Submodule

<CODE BEGINS> file "iana-geo-uri-type@2014-05-08.yang"

```
submodule iana-geo-uri-type {
    belongs-to "ietf-location" {
        prefix loc;
    }

    import ietf-yang-types {
        prefix yang;
    }

    organization "IANA";
    contact
        "
            Internet Assigned Numbers Authority

            Postal: ICANN
                4676 Admiralty Way, Suite 330
                Marina del Rey, CA 90292

            Tel:    +1 310 823 9358
            E-Mail: iana@iana.org";
    description
        "This YANG module defines the iana-geo-uri-type typedef,
        which contains YANG definitions for IANA-registered geo uri.

        This YANG module is maintained by IANA, serializes the existing
        'geo' URI into YANG format, and reflects the 'geo URI
        Parameters' registry and 'geo URI crs Parameter Values'
        sub-registry.

        The latest revision of this YANG module can be obtained from
        the IANA web site.

        Requests for new values should be made to IANA via
        email (iana@iana.org)."
```

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>).";

```

revision 2014-05-08 {
  description
    "Initial revision.";
}

typedef geo-uri {
  type yang:uri {
    pattern
      'geo:'
      + '-?0*(90|(((1-8)[0-9]|[0-9])(.[0-9]*)?))'
      + ',-?0*(180|((1[0-7][0-9]|[1-9][0-9]|[0-9])(.[0-9]*)?))'
      + '(-?[0-9]+(.[0-9]*)?)?'
      + '(;crs=(wgs84|[-0-9a-zA-Z]+))?'
      + '(;u=[0-9]+(.[0-9]*)?)?'
      + '([-da-zA-Z]+)'
      + '((([-_.!~*'()[]:&+$da-zA-Z]*)%[da-fA-F]{2}))*)*';
  }
  description
    "The geo-uri type represents a Uniform Resource Identifier for Geographic Locations ('geo' URI) as defined by RFC 5870.";
  reference
    "RFC 5870: A Uniform Resource Identifier for Geographic Locations";
}

```

#### 4. Location YANG module

```

<CODE BEGINS> file "ietf-location@2014-05-08.yang"

module ietf-location {
  namespace "urn:ietf:params:xml:ns:yang:ietf-location";
  prefix loc;

  include iana-civic-address-type {
    revision-date 2014-05-08;
  }
}

```

```
include iana-geo-uri-type {
  revision-date 2014-05-08;
}

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

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

  Author:     Yi Yang
              <mailto:yiya@cisco.com>

  Author:     Wojciech Dec
              <mailto:wdec@cisco.com>";

description
  "This YANG module defines a reusable group for location
  information";

revision 2014-05-08 {
  description
    "Initial revision.";
}

typedef country {
  type string {
    pattern '[A-Z]{2}';
  }
  description
    "ISO 3166 country code";
}

grouping location {
  container location {
    leaf geo-uri {
      type geo-uri;
      description
        "Uniform Resource Identifier for Geographic Locations
        ('geo' URI) as defined by RFC 5870.";
    }

    leaf country {
      type country;
      description
        "ISO 3166 country code";
    }
  }
}
```

```
leaf CA0 {
  type CA0;
  description
    "Language.";
}

leaf A1 {
  type A1;
  description
    "National subdivisions:
     state, canton, region, province, prefecture";
}

leaf A2 {
  type A2;
  description
    "county, parish, gun (JP), district (IN)";
}

leaf A3 {
  type A3;
  description
    "city, township, shi (JP)";
}

leaf A4 {
  type A4;
  description
    "city division, borough, city district, ward, chou (JP)";
}

leaf A5 {
  type A5;
  description
    "neighborhood, block";
}

leaf A6 {
  type A6;
  description
    "group of streets below the neighborhood level";
}

leaf PRD {
  type PRD;
  description
    "leading street direction";
}
```

```
leaf POD {
    type POD;
    description
        "trailing street suffix";
}

leaf STS {
    type STS;
    description
        "street suffix or type";
}

leaf HNO {
    type HNO;
    description
        "house number";
}

leaf HNS {
    type HNS;
    description
        "house number suffix";
}

leaf LMK {
    type LMK;
    description
        "landmark or vanity address";
}

leaf LOC {
    type LOC;
    description
        "additional location information";
}

leaf NAM {
    type NAM;
    description
        "name (residence and office occupant)";
}

leaf PC {
    type PC;
    description
        "postal/zip code";
}
```

```
leaf BLD {
  type BLD;
  description
    "building (structure)";
}

leaf UNIT {
  type UNIT;
  description
    "unit (apartment, suite)";
}

leaf FLR {
  type FLR;
  description
    "floor";
}

leaf ROOM {
  type ROOM;
  description
    "room number";
}

leaf PLC {
  type PLC;
  description
    "type of place";
}

leaf PCN {
  type PCN;
  description
    "postal community name";
}

leaf POBOX {
  type POBOX;
  description
    "post office box (P.O. Box)";
}

leaf ADDCODE {
  type ADDCODE;
  description
    "additional code";
}
```



```
leaf SEAT {
    type SEAT;
    description
        "seat (desk, cubicle, workstation)";
}

leaf RD {
    type RD;
    description
        "Primary road or street";
}

leaf RDSEC {
    type RDSEC;
    description
        "Road section";
}

leaf RDBR {
    type RDBR;
    description
        "Road branch";
}

leaf RDSUBBR {
    type RDSUBBR;
    description
        "Road sub-branch";
}

leaf PRM {
    type PRM;
    description
        "Street name pre-modifier";
}

leaf POM {
    type POM;
    description
        "Street name post-modifier";
}

leaf PN {
    type PN;
    description
        "Post number that is attributed to a lamp post or utility
        pole.";
}
```

```
    leaf MP {
      type MP;
      description
        "Milepost: a marker indicating distance to or from a place
        (often a town).";
    }

    leaf STP {
      type STP;
      description
        "Street Type Prefix.";
    }

    leaf HNP {
      type HNP;
      description
        "House Number Prefix.";
    }

    leaf STPS {
      type STPS;
      description
        "Separator between the existing Street Type Prefix (STP)
        and the Road Name (RD) elements.";
    }

    leaf LMKP {
      type LMKP;
      description
        "Component part of a Complete Landmark Name.";
    }

    leaf CA128 {
      type CA128;
      description
        "script";
    }
  }
}
```

## 5. Security Considerations

Since this document does not introduce any technology or protocol, there are no security issues to be considered for this document itself.

## 6. IANA Considerations

This document defines the initial version of the IANA-maintained iana-civic-address-type YANG submodule, and iana-geo-uri-type submodule.

The iana-civic-address-type submodule is intended to reflect the IANA "Civic Address Types Registry" [CATYPE-REGISTRY]. When an civic address type is added to the "Civic Address Types Registry", a new civic address data type must be added to the iana-civic-address-type submodule. When the iana-civic-address-type YANG submodule is updated, a new "revision" statement must be added in front of the existing revision statements.

<## do we need IANA to maintain the geo-uri submodule? ##>

## 6.1. URI Registrations

<TBD> This document needs to register a URI in the "IETF XML Registry" [XML-REGISTRY]. Following the format in [IETF-XML], the following registration needs to be made.

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

Registrant Contact: IANA.

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

## 6.2. YANG Module Registrations

This document registers a YANG module and two YANG submodules in the "YANG Module Names" registry [YANG-REGISTRY].

name:	ietf-location
namespace:	urn:ietf:params:xml:ns:yang:ietf-location
prefix:	loc
reference:	<TBD>

name:	iana-civic-address-type
module:	ietf-location
reference:	<TBD>

name:	iana-geo-uri-type
module:	ietf-location
reference:	<TBD>

## 7. Normative References

- [CATYPE-REGISTRY] Internet Assigned Numbers Authority, "Civic Address Types Registry", <<http://www.iana.org/assignments/civic-address-types-registry>>.
- [CRS-REGISTRY] Internet Assigned Numbers Authority, "'geo' URI 'crs' Parameter Values", <<http://www.iana.org/assignments/geo-uri-parameters/geo-uri-parameters.xhtml#geo-uri-parameters-2>>.
- [GEO-REGISTRY] Internet Assigned Numbers Authority, "'geo' URI Parameters", <<http://www.iana.org/assignments/geo-uri-parameters>>.
- [XML-REGISTRY] Internet Assigned Numbers Authority, "IETF XML Registry", <<http://www.iana.org/assignments/xml-registry>>.
- [YANG-REGISTRY] Internet Assigned Numbers Authority, "YANG Module Names", <<http://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml#yang-parameters-1>>.
- [CIVIC] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information", RFC 4776, November 2006.
- [GEOPRIV] Peterson, J., "A Presence-based GEOPRIV Location Object Format", RFC 4119, December 2005.
- [GEO-URI] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, June 2010.
- [IETF-XML] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [KEYWORD] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [YANG] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

#### Appendix A. Acknowledgments

The draft text was produced using Stefan Santesson's NroffEdit application.

Authors' Addresses

Yi Yang  
Cisco Systems  
7025 Kit Creek Road  
RTP, NC 27709  
USA

Email: [yiya@cisco.com](mailto:yiya@cisco.com)

Wojciech Dec  
Cisco Systems  
Haarlerbergpark  
Haarlerbergweg 13-19  
AMSTERDAM, NOORD-HOLLAND 1101 CH  
NETHERLANDS

Email: [wdec@cisco.com](mailto:wdec@cisco.com)

Internet  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2014

D. Yeung  
Y. Qu  
A. Clemm  
Cisco Systems  
October 22, 2013

Yang Data Model for OSPF Protocol  
draft-yeung-netmod-ospf-00

Abstract

This document defines a YANG data model that can be used to configure and manage OSPF.

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, 2014.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Overview . . . . .	2
1.1. Requirements Language . . . . .	2
2. Design of Data Model . . . . .	3
2.1. Overview . . . . .	3
2.2. OSPFv2 and OSPFv3 . . . . .	4
2.3. Optional Features . . . . .	4
2.4. Inheritance . . . . .	4
2.5. OSPF Router Configuration . . . . .	4
2.6. OSPF Address-Family Configuration . . . . .	5
2.7. OSPF Area Configuration . . . . .	6
2.8. OSPF Interface Configuration . . . . .	7
3. OSPF Yang Module . . . . .	9
4. Security Considerations . . . . .	29
5. Acknowledgements . . . . .	29
6. References . . . . .	30
6.1. Normative References . . . . .	30
6.2. Informative References . . . . .	30

## 1. Overview

YANG [RFC6020] is a data definition language that was introduced to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF [RFC6241]. YANG is proving relevant beyond its initial confines, as bindings to other interfaces (e.g. ReST) and encodings other than XML (e.g. JSON) are being defined. Furthermore, YANG data models can be used as the basis of implementation for other interfaces, such as CLI and programmatic APIs.

This document defines a YANG data model that can be used to configure and manage OSPF. Both OSPFv2 and OSPFv3 are supported. In addition to the core OSPF protocol, features described in different separate OSPF RFC are also supported. Those non-core features are made optional in the data model provided.

## 1.1. Requirements Language

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

## 2. Design of Data Model

Although the basis of OSPF configuration elements like routers, areas and interfaces remains the same, the detailed configuration model varies among different vendors. Differences are observed in term of how protocol engine is tied to routing domain, how multiple protocol engines could be instantiated and configuration inheritance, just to name a few.

The goal of this document is to define a data model that is capable of representing these differences. There is very little information that is designated as "mandatory", providing freedom to vendors to adapt this data model to their product implementation.

### 2.1. Overview

The OSPF YANG module defined in this document has all the common building blocks for OSPF protocol.

```
module: ospf
  +--rw ospf-routers
    +--rw ospf-router [version name]
      .
      .
    +--rw ospf-afs
      +--rw ospf-af [vrf-name afi safi]
        .
        .
      +--rw ospf-areas
        .
        .
      +--rw ospf-area [area-id]
        .
        .
      +--rw ospf-interfaces
        .
        .
      +--rw ospf-interface [interface]
        .
        .
```

The container ospf-routers allows specification of one or more OSPF configuration entities called ospf-router. The ospf-router is intended to match to the vendor specific OSPF configuration construct which is identified by a local identifier 'name'. The field 'version' allows support for OSPFv2 and OSPFv3.



The container `ospf-afs` includes one or more OSPF protocol engines, each encapsulated in the `ospf-af` entity. Each `ospf-af` includes information for the routing domain it is running on based on the `[vrf-name afi safi]` specification. There is no default routing domain assumed by the data model. For example, to enable OSPF on the default IPv4 routing domain of the vendor, this model requires an explicit `ospf-af` entity with the specification like `["default" "ipv4" "unicast"]`. The `ospf-af` also contains OSPF router level configuration

The `ospf-areas/ospf-area` and `ospf-intefaces/ospf-interface` contains the OSPF configuration for the area and interface level respectively

## 2.2. OSPFv2 and OSPFv3

The defined data model supports both OSPFv2 and OSPFv3.

The field 'version' is used to indicate the OSPF version and is a mandatory. Based on the version set, the data model change accordingly to accommodate the difference between the two versions.

## 2.3. Optional Features

Optional features are features beyond the basic of OSPF configurations and it is up to a vendor to decide the support of a particular feature on a particular device.

This module has declared a number of features, such as NSR, max-LSA etc.. It is intended that vendors will extend the features list.

## 2.4. Inheritance

This defined data model supports configuration inheritance for areas and interfaces.

Area related configurations specified in the `ospf-areas` container apply to all `ospf-area`.

Interface related configurations specified in `ospf-areas` container apply to all `ospf-interface` in all `ospf-area`'s.

Interface related configurations specified in `ospf-interfaces` container apply to all `ospf-interface` in the enclosing `ospf-area`.

## 2.5. OSPF Router Configuration

The container `ospf-routers` is the top level container in this data model. One or more `ospf-router` entity could be included in the `ospf-`

routers container. The main purpose of the ospf-router is to specify the OSPF version and the local name for the enclosed ospf-afs container which has the protocol configuration specifics.

```

module: ospf
  +--rw ospf-routers
    +--rw ospf-router [version name]
      +--rw version      uint8
      +--rw name         string
      .
      .

```

## 2.6. OSPF Address-Family Configuration

The container ospf-afs contains one or more ospf-af's each represents an OSPF protocol engine. Each ospf-af indicates the routing domain it is associated with based on [vrf-name afi safi] and contains the router level configurations.

```

module: ospf
  +--rw ospf-routers
    +--rw ospf-router [version name]
      ...
    +--rw ospf-afs
      +--rw ospf-af [vrf-name afi safi]
        +--rw vrf-name      string
        +--rw afi           enumeration
        +--rw safi          enumeration
        +--rw router-id
          | ...
        +--rw distance?      uint8
        +--rw default-originate
          | ...
        +--rw auto-cost
          | ...
        +--rw default-metric? uint32
        +--rw summary-prefix [prefix]
          | ...
        +--rw max-lsa
          | ...
        +--rw maximum
          | ...
        +--rw ignore
          | ...
        +--rw log
          | ...
        +--rw nsr

```

```

|   ...
+--rw nsf
|   ...
+--rw protocol
|   ...
+--rw spf
|   ...
+--rw snmp
|   ...
+--rw redistribute
|   ...
+--rw timers
|   ...
+--rw mpls
|   ...
+--rw fast-reroute
|   ...

```

## 2.7. OSPF Area Configuration

The container `ospf-areas` contains one or more `ospf-area`'s each represents an area in the OSPF protocol. Each `ospf-area` contains configurations of that area and the `ospf-interfaces` container which includes all the OSPF interfaces active in the enclosing area.

The `ospf-areas` also contains area configuration that could be inherited to all `ospf-area`'s defined. Similarly, the `ospf-area` also contains interface configuration that could be inherited to all the `ospf-interface`'s defined.

```

module: ospf
  +--rw ospf-routers
    +--rw ospf-router [version name]
      ...
    +--rw ospf-afs
      +--rw ospf-af [vrf-name afi safi]
        ...
      +--rw ospf-areas
        +--rw loopback?                               enumeration
        +--rw network-type?                             enumeration
        +--rw passive?                                   boolean
        +--rw mtu-ignore?                               boolean
        +--rw flood-reduction-enable?                   boolean
        +--rw demand-circuit?                           boolean
        +--rw demand-circuit-ignore-enable?             boolean
        +--rw prefix-suppression?                       boolean

```

```

+--rw ttl-security
...
+--rw packet-size
...
+--rw authentication
...
+--rw database-filter
...
+--rw neighbor [ip-address]
...
+--rw bfd
...
+--rw cost?                               uint16
+--rw hello-interval?                     uint16
+--rw dead-interval?                      uint16
+--rw retransmit-interval?                uint16
+--rw transmit-delay?                     uint16
+--rw ospf-area [area-id]
    +--rw area-id                         uint32
    +--rw area-notation?                  enumeration
    +--rw (area-type)?
        ...
    +--rw default-cost?                    uint32
    +--rw range [prefix]
        ...
    +--rw virtual-link
        ...
    +--rw sham-link
        ...
+--rw mpls-te-config
...

```

## 2.8. OSPF Interface Configuration

The container `ospf-areas` contains one or more `ospf-interface`'s each represents an interface in the OSPF protocol. Each `ospf-interface` contains configurations of that interface.

The `ospf-interfaces` also contain interface configuration that could be inherited to all `ospf-interface`'s defined.

```

module: ospf
  +--rw ospf-routers
    +--rw ospf-router [version name]
      ...
    +--rw ospf-afs
      +--rw ospf-af [vrf-name afi safi]

```

```

...
+--rw ospf-areas
    ...
    +--rw ospf-interfaces
        +--rw network-type?          enumeration
        +--rw passive?               boolean
        +--rw mtu-ignore?            boolean
        +--rw flood-reduction-enable? boolean
        +--rw demand-circuit?        boolean
        +--rw demand-circuit-ignore-enable? boolean
        +--rw prefix-suppression?    boolean
        +--rw ttl-security
            ...
        +--rw packet-size
            ...
        +--rw authentication
            ...
        +--rw database-filter
            ...
        +--rw neighbor [ip-address]
            ...
        +--rw bfd
            ...
        +--rw cost?                  uint16
        +--rw hello-interval?       uint16
        +--rw dead-interval?        uint16
        +--rw retransmit-interval?   uint16
        +--rw transmit-delay?        uint16
        +--rw ospf-interface [interface]
            +--rw interface          if:interface-ref
            +--rw network-type?      enumeration
            +--rw passive?           boolean
            +--rw mtu-ignore?        boolean
            +--rw flood-reduction-enable? boolean
            +--rw demand-circuit?    boolean
            +--rw demand-circuit-ignore-enable? boolean
            +--rw prefix-suppression? boolean
            +--rw ttl-security
                ...
            +--rw packet-size
                ...
            +--rw authentication
                ...
            +--rw database-filter
                ...
            +--rw neighbor [ip-address]
                ...
            +--rw bfd

```

```

    ...
    +--rw cost?                               uint16
    +--rw hello-interval?                     uint16
    +--rw dead-interval?                      uint16
    +--rw retransmit-interval?                uint16
    +--rw transmit-delay?                     uint16

```

### 3. OSPF Yang Module

```

<CODE BEGINS>
module ospf {
  namespace "urn:cisco:params:xml:ns:yang:ospf";
  // replace with IANA namespace when assigned
  prefix ospf;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-interfaces {
    prefix if;
  }

  organization
    "Cisco Systems
     170 West Tasman Drive
     San Jose, CA 95134-1706
     USA";
  contact
    "Derek Yeung myeung@cisco.com
     Yingzhen Qu yiqu@cisco.com
     Alexander Clemm alex@cisco.com";

  description
    "This YANG module defines the generic configuration
     data for OSPF, which is common across all of the vendor
     implementations of the protocol. It is intended that the module
     will be extended by vendors to define vendor-specific
     OSPF configuration parameters and policies,
     for example route maps or route policies.

     Terms and Acronyms

     OSPF (ospf): Open Shortest Path First

     IP (ip): Internet Protocol

```

```
    IPv4 (ipv4):Internet Protocol Version 4

    IPv6 (ipv6): Internet Protocol Version 6

    MTU (mtu) Maximum Transmission Unit
    ";

revision 2013-10-20 {
    description
        "Initial revision.";
}

feature flood-reduction {
    description
        "OSPF Flood Reduction.";
}

feature demand-circuit-ignore {
    description
        "Ignore demand circuit auto-negotiation requests.";
}

feature security {
    description
        "Enable/Disable TTL security.";
}

feature packet-size {
    description
        "Customize size of OSPF packets up to MTU.";
}

feature nsf-cisco {
    description
        "Enable Cisco Non Stop Forwarding.";
}

feature loopback-config {
    description
        "OSPF loopback configuration.";
}

feature max-lsa {
    description
        "Maximum number of LSAs OSPF process will receive.";
}

feature maximum-ecmp {
```

```
    description
      "Maximum number of ECMP paths.";
  }

  feature nsr {
    description
      "Enable NSR.";
  }

  feature spf {
    description
      "SPF configuration";
  }

  feature nssa-only {
    description
      "Limit summary to NSSA areas.";
  }

  feature nssa-ext-capability {
    description
      "Send domain specific capabilities into NSSA.";
  }

  grouping interface-internal-inherit-config {
    leaf cost {
      description
        "Interface cost.";
      type uint16 {
        range "1..65535";
      }
    }

    leaf hello-interval {
      description
        "Time between HELLO packets.";
      units seconds;
      type uint16 {
        range "1..65535";
      }
    }

    leaf dead-interval {
      description
        "Interval after which a neighbor is declared dead.";
      units seconds;
      type uint16 {
        range "1..65535";
      }
    }
  }
```



```
    }
    must "dead-interval > ../hello-interval" {
        error-app-tag dead-interval-invalid;
        error-message "The dead interval must be "
            + "larger than the hello interval";
    }
}

leaf retransmit-interval {
    description
        "Time between retransmitting lost link state
        advertisements.";
    units seconds;
    type uint16 {
        range "1..65535";
    }
}

leaf transmit-delay {
    description
        "Estimated time needed to send link-state update.";
    units seconds;
    type uint16 {
        range "1..65535";
    }
}
} // interface-internal-inherit-config

grouping interface-inherit-config {
    leaf network-type {
        description
            "Network type.";
        type enumeration {
            enum "broadcast" {
                description
                    "Specify OSPF broadcast multi-access network.";
            }
            enum "non-broadcast" {
                description
                    "Specify OSPF NBMA network.";
            }
            enum "point-to-multipoint" {
                description
                    "Specify OSPF point-to-multipoint network.";
            }
            enum "point-to-point" {
                description
                    "Specify OSPF point-to-point network.";
            }
        }
    }
}
```

```
    }  
  }  
}  
  
leaf passive {  
  description  
    "Enable/Disable passive.";  
  type boolean;  
}  
  
leaf mtu-ignore {  
  description  
    "Enable/Disable ignoring of MTU in DBD packets.";  
  type boolean;  
}  
  
leaf flood-reduction-enable {  
  if-feature flood-reduction;  
  type boolean;  
}  
  
leaf demand-circuit {  
  description  
    "Enable/Disable demand circuits.";  
  type boolean;  
}  
  
leaf demand-circuit-ignore-enable {  
  if-feature demand-circuit-ignore;  
  type boolean;  
}  
  
leaf prefix-suppression {  
  description  
    "Suppress advertisement of the prefixes.";  
  type boolean;  
}  
  
container ttl-security {  
  if-feature security;  
  leaf enable {  
    type boolean;  
  }  
  leaf hops {  
    type uint8 {  
      range "1..254";  
    }  
  }  
}
```

```
}

container packet-size {
  if-feature packet-size;
  description
    "Customize size of OSPF packets up to MTU.";
  leaf enable {
    type boolean;
  }
  leaf size {
    type uint16 {
      range "576..10000";
    }
  }
}

container authentication {
  when "../../version = 2";
  description "Configure authentication.";
  leaf enable {
    type boolean;
  }
  choice authentication-type-selection {
    case clear-text {
      description
        "Use clear-text authentication.";
      leaf key {
        type string {
          length "1..8";
        }
      }
    }
    case message-digest {
      description
        "Use message-digest authentication.";
      list message-digest-key {
        key key-id;
        leaf key-id {
          type uint8 {
            range "1..255";
          }
        }
        leaf algorithm {
          type enumeration {
            enum "md5";
          }
        }
        leaf key {

```

```
        type string {
            length "1..8";
        }
    }
}

container database-filter {
    description
        "Filter OSPF LSA during synchronization and flooding";
    leaf enable {
        type boolean;
    }
    leaf lsa-type {
        type enumeration {
            enum "all";
        }
    }
    leaf direction {
        type enumeration {
            enum "out";
        }
    }
}

list neighbor {
    description
        "Specify a neighbor router.";
    key ip-address;
    leaf ip-address {
        type inet:ip-address;
    }
    leaf cost {
        type uint16 {
            range "1..65535";
        }
    }
    leaf poll-interval {
        units seconds;
        type uint16 {
            range "1..65535";
        }
    }
    leaf priority {
        type uint8 {
            range "1..255";
        }
    }
}
```

```
    }  
  }  
}  
  
container bfd {  
  leaf enable {  
    description  
      "enable bfd.";  
    type boolean;  
  }  
  
  leaf minimum-interval {  
    description  
      "bfd hello interval";  
    type uint16 {  
      range "15..30000";  
    }  
  }  
  leaf multiplier {  
    description  
      "detect multiplier";  
    type uint8 {  
      range "2..50";  
    }  
  }  
}  
  
uses interface-internal-inherit-config;  
  
} // grouping interface-inherit-config  
  
grouping area-inherit-config {  
  leaf loopback {  
    if-feature loopback-config;  
    description  
      "OSPF loopback configuration.";  
    type enumeration {  
      enum "stub";  
      enum "host";  
    }  
  }  
}  
  
} // grouping area-inherit-config  
  
container ospf-routers {  
  list ospf-router {  
    description  
      "This is a top-level container for the OSPF router.";  
  }  
}
```

```
key "version name";
leaf version {
  description
    "OSPF version.";
  mandatory true;
  type uint8 {
    range "2..3";
  }
}

leaf name {
  description
    "Name, combined with
    ospf-routers/ospf_router/ospf-afs/ospf-af,
    identify an OSPF protocol instance.";
  mandatory true;
  type string;
}

container ospf-afs {
  list ospf-af {
    key "vrf-name afi safi";
    leaf vrf-name {
      type string;
    }
    leaf afi {
      type enumeration {
        enum "ipv4" {
          description
            "IPv4";
        }
        enum "ipv6" {
          description
            "IPv6";
        }
      }
    }
    leaf safi {
      type enumeration {
        enum "unicast" {
          description
            "unicast";
        }
      }
    }
  }

  container router-id {
    description
```

```
        "Specify the router-id for this OSPF process.";
    leaf enable {
        type boolean;
    }
    choice config-type {
        case static {
            leaf ip-address {
                type boolean;
            }
        }
        case auto-config {
            leaf enable-auto-config {
                type boolean;
            }
        }
    }
}

leaf distance {
    description
        "Define an administrative distance.";
    type uint8 {
        range "1..255";
    }
}

container default-originate {
    description
        "Control distribution of default route information.";
    leaf enable {
        type boolean;
        default "false";
    }
}

container auto-cost {
    description
        "Calculate OSPF interface cost according to
        bandwidth.";
    leaf enable {
        type boolean;
    }
    leaf reference-bandwidth {
        type uint32 {
            range "1..4294967";
        }
    }
}
```

```
leaf default-metric {
  description
    "Calculate OSPF interface cost according to
    bandwidth.";
  type uint32 {
    range "1..16777214";
  }
}

list summary-prefix {
  description
    "Configure IP address summaries";
  key "prefix";
  leaf prefix {
    type inet:ip-prefix;
  }
  leaf advertise {
    type boolean;
  }
  leaf tag {
    type uint32 {
      range "0..4294967295";
    }
  }
  leaf nssa-only-enable {
    if-feature nssa-only;
    type boolean;
  }
}

container max-lsa {
  if-feature max-lsa;
  description
    "Maximum number of LSAs OSPF process will
    receive.";
  leaf max {
    description
      "Maximum number of non self-generated LSAs
      this process can receive.";
    type uint32 {
      range "1..4294967294";
    }
  }
  leaf threshold {
    description
      "Threshold value (%) at which to generate
      a warning msg.";
    type uint8 {
```



```
        range "1..100";
    }
}
leaf warning-only {
    description
        "Only give warning message when limit is
        exceeded.";
    type boolean;
}
leaf ignore-time {
    description
        "Number of minutes during which all adjacencies
        are suppressed.";
    type uint16 {
        range "1..17895";
    }
}
leaf ignore-count {
    description
        "Count on how many times adjacencies can be
        suppressed.";
    type uint16 {
        range "1..65534";
    }
}
leaf reset-time {
    description
        "number of minutes after which ignore-count is
        reset to zero.";
    type uint16 {
        range "2..35791";
    }
}
}

container maximum {
    if-feature maximum-ecmp;
    description
        "Set OSPF limits.";
    leaf paths {
        description
            "Maximum number of ECMP paths.";
        type uint16 {
            range "1..32";
        }
    }
    leaf interfaces {
        description
```

```
        "Maximum number of interfaces.";
    type uint16 {
        range "1..1024";
    }
}
container redistributed-prefixes {
    description
        "Maximum number of prefixes redistributed.";
    leaf max {
        type uint32 {
            range "1..4294967295";
        }
    }
    leaf threshold {
        description
            "Threshold value (%) at which to generate
             a warning msg.";
        type uint8 {
            range "1..100";
        }
    }
    leaf warning-only {
        description
            "Only give warning message when limit is
             exceeded.";
        type boolean;
    }
}

container ignore {
    description
        "Do not complain about specific event.";
    leaf-list lsa {
        type enumeration {
            enum "mospf";
        }
    }
}

container log {
    description
        "Log ospf info.";
    leaf adjacency-changes {
        type boolean;
    }
}
```

```
    container nsr {
      if-feature nsr;
      description
        "Enable NSR.";
      leaf enable {
        type boolean;
      }
    }

    container nsf {
      choice nsf-type {
        case cisco {
          if-feature nsf-cisco;
          description
            "Enable Cisco Non Stop Forwarding.";
          leaf cisco-enable {
            type boolean;
          }
        }
        case ietf {
          description
            "Enable Cisco Non Stop Forwarding.";
          leaf ietf-enable {
            type boolean;
          }
          leaf ietf-helper-enable {
            type boolean;
          }
        }
      }
    }

    container protocol {
      description
        "Protocol specific configuration.";
      leaf shutdown {
        type boolean;
      }
    }

    container spf {
      if-feature spf;
      description
        "SPF configuration";
      leaf prefix-priority-policy {
        type string;
      }
    }
  }
```

```
    container snmp {
      description
        "Adjust ospf snmp parameters";
      leaf context {
        type string;
      }
    }

    container redistribute {
      description
        "Redistribute information from another
        routing protocol.
        This grouping is intended to be augmented by
        vendors to implement vendor-specific protocol
        redistribution configuration options.";
      choice protocol {
        case bgp {
          leaf enable-bgp {
            type boolean;
          }
        }
        case ospf {
          leaf enable-ospf {
            type boolean;
          }
        }
        case isis {
          leaf enable-isis {
            type boolean;
          }
        }
        case connected {
          leaf enable-connected {
            type boolean;
          }
        }
        case eigrp {
          leaf enable-eigrp {
            type boolean;
          }
        }
        case mobile {
          leaf enable-mobile {
            type boolean;
          }
        }
        case static {
          leaf enable-static {
```

```
        type boolean;
    }
}
case rip {
    leaf enable-rip {
        type boolean;
    }
}
}
}

container timers {
    description
        "Adjust routing timers.
        This grouping is intended to be augmented
        by vendors to implement vendor-specific
        protocol timers configuration options";

    container lsa {
        leaf min-arrival {
            description
                "The minimum interval in milliseconds
                between accepting the same";
            units milliseconds;
            type uint32 {
                range "0..600000";
            }
        }
        leaf refresh {
            description
                "The minimum interval in seconds
                between refresh";
            units seconds;
            type uint32 {
                range "1800..2700";
            }
        }
    }
}

container throttle-lsa {
    leaf delay {
        description
            "Delay to generate first occurrence of
            LSA in milliseconds";
        units milliseconds;
        type uint32 {
            range "0..600000";
        }
    }
}
```

```
leaf min-delay {
  description
    "The Minimum delay between originating
    the same LSA in milliseconds";
  units milliseconds;
  type uint32 {
    range "0..600000";
  }
}
leaf max-delay {
  description
    "The Maximum delay between originating
    the same LSA in milliseconds";
  units milliseconds;
  type uint32 {
    range "0..600000";
  }
}
}

container throttle-spf {
  leaf delay {
    description
      "Delay between receiving a change to
      SPF calculation in milliseconds";
    units milliseconds;
    type uint32 {
      range "0..600000";
    }
  }
  leaf min-delay {
    description
      "Delay between first and second SPF
      calculation in milliseconds";
    units milliseconds;
    type uint32 {
      range "0..600000";
    }
  }
  leaf max-delay {
    description
      "Maximum wait time in milliseconds
      for SPF calculations";
    units milliseconds;
    type uint32 {
      range "0..600000";
    }
  }
}
```

```
    }  
  }  
  
  container mpls {  
    description  
      "OSPF MPLS configuraions.";  
    leaf te-rid {  
      description  
        "Traffic Engineering stable  
         IP address for system.";  
      type if:interface-ref;  
    }  
    leaf ldp-sync-enable {  
      description  
        "Enable LDP IGP synchronization.";  
      type boolean;  
    }  
    leaf ldp-autoconfig-enable {  
      description  
        "Enable LDP IGP interface  
         auto-configuration.";  
      type boolean;  
    }  
  }  
}  
  
container fast-reroute {  
  leaf fast-reroute-enable {  
    description  
      "IP fast reroute.";  
    type boolean;  
  }  
}  
  
container ospf-areas {  
  description  
    "The top level container for the list  
     of areas of the OSPF router.";  
  
  uses area-inherit-config;  
  uses interface-inherit-config;  
  
  list ospf-area {  
    key "area-id";  
    leaf area-id {  
      type uint32;  
    }  
  
    leaf area-notation {
```

```
description
  "Sets the default area notation.";
type enumeration {
  enum "decimal";
  enum "dot";
}
}
choice area-type {
  case nssa {
    description
      "Specify area as a NSSA area.";
    leaf redistribute {
      type boolean;
    }
    leaf nssa-summary {
      type boolean;
    }
    leaf nssa-ext-capability {
      if-feature nssa-ext-capability;
      type boolean;
    }
  }
  container default-information-originate {
    description
      "Originate Type 7 default into NSSA area.";
    leaf metric {
      type uint16 {
        range "1..65535";
      }
    }
    leaf metric-type {
      type uint8 {
        range "1..2";
      }
    }
  }
}
}
case stub {
  description
    "Specify area as a stub area.";
  leaf stub-summary {
    type boolean;
  }
  leaf stub-ext-capability {
    type boolean;
  }
}
}
```



```
leaf default-cost {
  description
    "Set the summary default-cost of a
    NSSA/stub area.";
  type uint32 {
    range "1..16777215";
  }
}

list range {
  description
    "Summarize routes matching
    address/mask (border routers only)";
  key "prefix";
  leaf prefix {
    type inet:ip-prefix;
  }
  leaf advertise {
    type boolean;
  }
  leaf cost {
    type uint32 {
      range "0..16777214";
    }
  }
}

container virtual-link {
  description
    "Define a virtual link";
  leaf router-id {
    type inet:ip-address;
  }

  uses interface-internal-inherit-config;
}

container sham-link {
  leaf local-id {
    description
      "Address of the local end-point.";
    type inet:ip-address;
  }
  leaf remote-id {
    description
      "Address of the remote end-point.";
    type inet:ip-address;
  }
}
```

```
        uses interface-internal-inherit-config;
    }

    container mpls-te-config {
        leaf mpls-te-enable {
            description
                "Enable an ospf area to run MPLS
                Traffic Engineering.";
            type boolean;
        }
    }

    container ospf-interfaces {
        description
            "The top level container for the
            list of interfaces of the OSPF router.";

        uses interface-inherit-config;

        list ospf-interface {
            key "interface";
            leaf interface {
                type if:interface-ref;
            }
            uses interface-inherit-config;
        } // list ospf-interface
    } // container ospf-interfaces
} // list ospf-area
} // container ospf-areas
} // container ospf-af
} // container ospf-afs
} // list ospf-router
} // container ospf-routers
}
<CODE ENDS>
```

#### 4. Security Considerations

The data model defined does not create any security implications.

This draft does not change any underlying security issues inherent in [I-D.ietf-netmod-routing-cfg].

#### 5. Acknowledgements

The authors would like to thank Gaurav Gupta and many others for review and comments.

This document was produced using Marshall Rose's xml2rfc tool.

## 6. References

### 6.1. 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.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

### 6.2. Informative References

- [I-D.ietf-netmod-interfaces-cfg]  
Bjorklund, M., "A YANG Data Model for Interface Management", draft-ietf-netmod-interfaces-cfg-12 (work in progress), July 2013.
- [I-D.ietf-netmod-routing-cfg]  
Lhotka, L., "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-11 (work in progress), October 2013.

## Authors' Addresses

Derek Yeung  
Cisco Systems  
170 West Tasman Drive  
San Jose, CA 95134  
USA

EMail: [myeung@cisco.com](mailto:myeung@cisco.com)

Yingzhen Qu  
Cisco Systems  
170 West Tasman Drive  
San Jose, CA 95134  
USA

EMail: [yiqu@cisco.com](mailto:yiqu@cisco.com)

Alexander Clemm  
Cisco Systems  
170 West Tasman Drive  
San Jose, CA 95134  
USA

EMail: alex@cisco.com

Network Working Group  
Internet Draft  
Intended status: Standard Track  
Expires: January 3, 2015

G. Zheng  
X. Ji  
Huawei Technologies  
July 2, 2014

Integrating Operations in YANG Models  
draft-zheng-netmod-integrate-operations-00.txt

## Abstract

This document introduces an extension to YANG. The extension allows operation methods to be directly integrated in YANG models.

## 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 January 3, 2015.

## Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction .....	3
2. Requirements Language and Terminology .....	3
3. Scenarios and Requirements of Operation Extension in Data Models	3
3.1. Scenarios of Complex Operations .....	3
3.2. Scenario of Operation Authorization .....	6
3.3. The Requirements for Operation Extension in Data Models ..	6
4. Solution for Operation Extension in Data Models .....	6
4.1. General Idea .....	6
4.2. Usage Example .....	7
5. Security Considerations .....	9
6. IANA Considerations .....	9
7. References .....	9
7.1. Normative References .....	9
7.2. Informative References .....	10
8. Acknowledgments .....	10
Authors' Addresses .....	11

## 1. Introduction

YANG is a data modeling language used to model data manipulated by the NETCONF Protocol. YANG provides the capability to describe configuration/state data in a tree structure; and it has been proven that YANG is an efficient language to model data for network configuration.

However, along with management requirements of modern NMS becoming more and more sophisticate, the capability of only describing the data structure in the model seems to be insufficient in some scenarios. In a nutshell, these scenarios require some operations to be integrated in the model as well.

This document firstly describes the scenarios and then introduces a YANG model extension which allows operation method to be directly integrated in YANG models.

## 2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Terminology:

## 3. Scenarios and Requirements of Operation Extension in Data Models

In some datastore operation scenarios, the operations triggered from the NETCONF client might be a set of combined operations or a complex single operation within the NETCONF agent, which may change the datastores. And sometimes the datastore change might not be from client but from the agent itself. These operations need to manipulate the data in running datastore or candidate datastore, and thus they need the <edit-config> capability as well as a unified authorization system to control the influence to the data stores caused by them.

### 3.1. Scenarios of Complex Operations

The complex operations on data model such as batch creating or converting dynamic data to configuration data, are targeting the

running datastore or candidate datastore. Thus, the target datastore, error-option, confirmed-commit, validation and other attributes of <edit-config> operation need to be specified. The following are some examples.

o Scenario 1: converting the device's dynamic ARP to static ARP

The ARP data model contains static configuration attributes and dynamic non-configuration attributes; both of them are needed by ARP service. The difference between these two groups of attributes is that the static ARP is configured by the client while the dynamic ARP is learned from network peers. To accelerate the speed of dynamic ARP learning after rebooting device, or to simplify the configuration of the static ARP, the dynamic ARP entries need to be converted to static ARP entries when the network is stable.

Current solution 1:

The client gets the dynamic ARP data then uses these data as static ARP configuration and add them to the NETCONF agent by using <edit-config> operation.

Problems of solution 1:

- There are too many data communications between client and server, thus the performance is low.
- The configured ARP by dynamic ARP from client is duplicated with the dynamic ARP in the device, thus the device has to process the duplication.

Current solution 2:

According to [RFC6020]/[RFC6241] RPC definition, define a new non-standard RPC to convert dynamic ARP to static ARP directly.

Problems of solution 2:

The new RPC cannot use the existing attributes of <edit-config> operation, such as target datastore, error-option, confirmed commit and validate. The NETCONF agent has to implement these <edit-config> operation attributes itself, which makes the agent more complex.

o Scenario 2: batch operations on data model



In some situations, batch merge operations on the list of one module might be needed to reduce the communication data size.

Current solutions:

- 1) Use <get-config> with filter to get the data from the device, and then add all merge operations to <edit-config> and send it to the device.
- 2) According to [RFC6020]/[RFC6241] RPC definition, define a non-standard RPC to do the batch merge operation.

Problems of current solutions

- 1) Solution 1 impacts the operation efficiency between host and device significantly.
- 2) Solution 2 has the same issues with Use Case 1's solution 2.

o Scenario 3: maintaining operations on data models

The data model of one module is usually hierarchical. Sometimes it needs maintaining operations on the sub nodes, such as restarting the running services.

Current solution:

According to [RFC6020]/[RFC6241] RPC definition, define a non-standard RPC to do maintaining operations.

Problems of current solution:

- 1) When defining the RPC operation, all the key attributes of parents path should appear in the input parameter of the RPC, otherwise the instance of the sub node could not be located. The additional key parameters make the input parameter definition redundant.
- 2) The scope of sub nodes that the new defined RPC would impact is not clear, thus users have to refer other documents to do authorization rather than just referring the RPC definition itself.

### 3.2. Scenario of Operation Authorization

The user-defined non-standard RPC operations might impact the datastores, which need additional authorization.

Current solution:

When the user-defined RPC operation impact the datastores (either directly manipulate the datastores or impact the datastores as a side effect of the protocol operation), then the server **MUST** intercept the access operation and make sure the user is authorized to perform the requested access operation on the targeted data as defined in [RFC6536] Section 3.4.5.

Problems of Current Solution:

1) Because the scope of datastores that the RPC operation would impact is not defined exactly, there is a risk that invocation of non-standard protocol operations might have undocumented side effects ([RFC6536] Section 3.7.2).

2) An administrator needs to set access control rules to make the configuration datastore protected from user-defined non-standard protocol operation's side effects. But, the rules definition has to depend on the relevant documents not directly on model definition, thus it is not the same as the data node access control rule definition, which makes the authorization management more complex.

### 3.3. The Requirements for Operation Extension in Data Models

The user-defined non-standard operation should overload the standard operations of <edit-config>, <get>, <get-config> and so on, so that the new operation could inherit the capabilities of the base standard operations.

The user-defined non-standard operations should be able to clearly define the scope of the datastores that the operations are allowed to impact in terms of build-in RPC semantics rather than outside documents.

## 4. Solution for Operation Extension in Data Models

### 4.1. General Idea

1) The operations to data nodes should be defined on the data model hierarchy tree as a part of the data model attributes; the data

node's sub-statement of the operations should also be defined as well..

2) The operations that will effect or side effect datastores should be specified which base standard RPC operations are inherited.

3) When the data model operations are delivered by NETONCF, they MUST be in the format of the base protocol RPC operations; and the operations definition should be presented as a "Method" attribute of data node; the base protocol RPC operation capabilities should be inherited.

4) The operation access control rules should depend on the impact to data models by the operations defined in the model.

#### 4.2. Usage Example

This example is based on the ARP conversion scenario described in Section 3.1.

Operation is defined as a "method" substatement in the YANG model:

```
module arp {  
    namespace "http://example.com/network";  
    prefix "arp";  
    container arp-records {  
        list arpList {  
            leaf ipAddr {  
                type string  
            }  
            leaf macAddr {  
                type string  
            }  
            leaf styleType {  
                type string;  
            }  
        }  
    }  
}
```

```
    }  
    method convert-arp {  
        base edit-config  
  
        description "A method to convert dynamic arp to static  
        arp";  
  
        input {  
            leaf source-arptype {  
                type string;  
            }  
            leaf dest-arptype {  
                type string;  
            }  
        }  
    }  
}
```

Corresponding NETCONF operation is as the following:

```
<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>
  <arp xmlns="http://example.com/network">
    <arp-records method="convert-arp">
      <source-arptype>dynamic</source-arptype >
      <dest-arptype>dynamic</dest-arptype >
    </arp-records >
  </arp>
</config>
</edit-config>
</rpc>
```

## 5. Security Considerations

TBD

## 6. IANA Considerations

This document requires no IANA registration.

## 7. References

### 7.1. 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.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, July 2013.

## 7.2. Informative References

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

## 8. Acknowledgments

Many valuable comments were received from Gang Yan to improve the draft. Bing Liu also participated in forming this draft.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Guangying Zheng  
Huawei Nanjing R&D Center  
Huawei Technologies Co., Ltd  
101 Software Avenue, Yuhua District, Nanjing, Jiangsu  
P.R.China. 210012

Email: zhengguangying@huawei.com

Xiaofeng Ji  
Q14-4-B Building  
Huawei Technologies Co., Ltd  
Zhong-Guan-Cun Environmental Protection Park, No.156 Beiqing Rd.  
Hai-Dian District, Beijing  
P.R. China. 100095

Email: jixiaofeng@huawei.com

