

NETMOD
Internet-Draft
Intended status: Informational
Expires: April 19, 2016

D. Bogdanovic
B. Claise
C. Moberg
Cisco Systems, Inc.
October 17, 2015

YANG Model Classification
draft-bogdanovic-netmod-yang-model-classification-05

Abstract

The YANG [RFC6020] data modeling language is currently being considered for a wide variety of applications throughout the networking industry at large. Many standards defining organizations, open source projects, and vendors are using YANG to develop and publish models of configuration, state data and operations for a wide variety of applications. At the same time, there is currently no well-known terminology to categorize various types of YANG models.

A consistent terminology would help with the categorization of models, assist in the analysis the YANG data modeling efforts in the IETF and other organizations, and bring clarity to the YANG-related discussions between the different groups.

This document describes a set of concepts and associated terms to support consistent classification of 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 April 19, 2016.

Copyright Notice

Copyright (c) 2015 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. First Dimension: YANG Model Abstraction Layers	3
2.1. Network Service YANG Data Models	4
2.2. Network Element YANG Data models	5
3. Second Dimension: Model Types	6
3.1. Standard YANG Models	6
3.2. Vendor-specific YANG Models	6
3.3. Vendor-specific Extensions	7
4. Security Considerations	7
5. IANA Considerations	7
6. Acknowledgements	7
7. Change log [RFC Editor: Please remove]	7
8. References	7
8.1. Normative References	8
8.2. Informative References	8
Authors' Addresses	8

1. Introduction

The Internet Engineering Steering Group (IESG) has been actively encouraging IETF working groups to use the NETCONF [RFC6241] and YANG standards for configuration management purposes, especially in new working group charters [Writable-MIB-Module-IESG-Statement].

YANG is also gaining wide acceptance as the de-facto standard modeling language in the broader industry. This extends beyond the IETF, including many standards development organizations, industry consortia, ad hoc groups, open source projects and vendors.

There are currently no clear guidelines on how to classify the layering of YANG models according to abstraction, or how to classify

models along the continuum spanning formal standards publications and vendor-specific models.

This document presents a set of concepts and terms to form a useful taxonomy for consistent classification of YANG models in two dimensions:

- o The layering of models based on their abstraction levels
- o The type of model based on the nature and intent of the content

The two categories are covered in the next two sections.

2. First Dimension: YANG Model Abstraction Layers

Model developers have taken two approaches to developing YANG model: top-down and bottom-up. The top-down approach starts with high level abstractions modeling business or customer requirements and maps them to specific networking technologies. The bottom-up approach starts with fundamental networking technologies and maps them into more abstract constructs.

There are currently no specific requirements on, or well-defined best practices around the development of models. For the purpose of this document we assume that both approaches (bottom-up and top-down) will be used as they both provide benefits that appeals to different groups.

For layering purposes, this documents suggests the classification of data models into two distinct abstraction layers:

- o Network Element YANG Models describe the configuration, state data and operations of a specific device-centric technology or feature.
- o Network Service YANG Models describes the configuration, state data and operations of an abstract representation of a service implemented on one or multiple network elements

Figure 1 illustrates the application of YANG models at different layers of abstraction. Layering of models allow for reusability of existing lower layer models by higher level models while limiting duplication of features across layers.

For model developers, per-layer modeling allows for separation of concern across editing teams focusing on specific areas.

As an example, experience from the IETF shows that creating useful network element YANG models for e.g routing or switching protocols

requires teams that include developers with experience from implementing those protocols.

On the other hand, network service models are best developed by people experienced in defining network services for consumption by programmers developing e.g. flow-through provisioning systems or self-service portals.

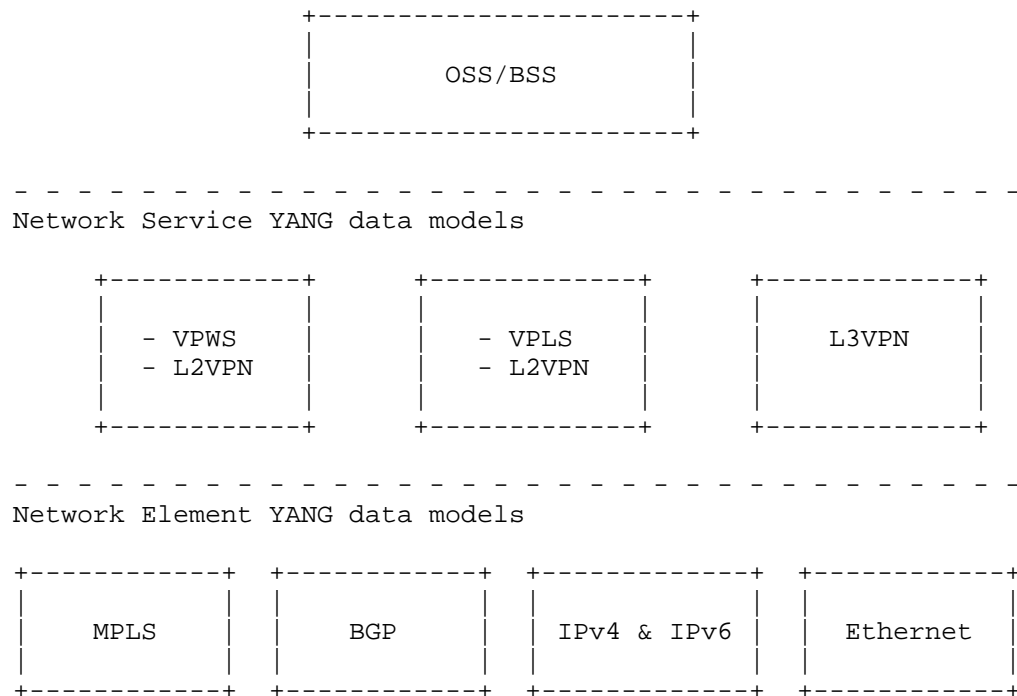


Fig. 1 YANG Model Layers

2.1. Network Service YANG Data Models

Network Service YANG Data Models describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service model does not expose the detailed configuration parameters of all participating network elements and features, but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element data models of the participating network elements. The service-to-element decomposition is a separate process with details depending on how the network operator chooses to realize the service.

As an example, the Network Service model included in [YANG-Data-Model-for-L3VPN-service-delivery] provides an abstracted model for Layer 3 IP VPN service configuration. An orchestrator receives operations on service instances according to the service model and decomposes the data into specific Network Element models to configure the participating network elements to perform the intent of the service.

Network Service YANG models defines services models to be consumed by external systems. These models are commonly designed, developed and deployed by network infrastructure teams.

YANG allows for different design patterns to describe network services, ranging from monolithic to component-based approaches.

The monolithic approach captures the entire service in a single model and does not put focus on reusability of internal data definitions and groupings. The monolithic approach has the advantages of single-purpose development including speed at the expense of reusability.

The component-based approach captures device-centric features (e.g. the definition of a VRF, routing protocols, or packet filtering) in a vendor-independent manner. The components are designed for reuse across many services. The set of components required for a specific service is then composed into the higher-level service. The component-based approach has the advantages of modular development including a higher degree of reusability at the expense of initial speed.

As an example, an L2VPN service can be built on many different types of transport network technologies, including e.g. MPLS or carrier ethernet. A component-based approach would allow for reuse of e.g. UNI-interface definitions independent of the underlying transport network (e.g. MEF UNI interface or MPLS interface). The monolithic approach would assume a specific set of transport technologies and interface definitions.

2.2. Network Element YANG Data models

Network Element YANG Data Models describe the configuration, state data and operations of a network device as defined by the vendor of that device. The models are commonly structured around features of the device, e.g. interface configuration [RFC7223], OSPF configuration [I-D.ietf-ospf-yang], and firewall rules definitions [I-D.ietf-netmod-acl-model]. The model provides a coherent data model representation of what is commonly a very mixed software environment consisting of the operating system and applications running on the device.

The decomposition, ordering and execution of changes to the operating system, and application configuration is the task of the management framework that implements the YANG model.

3. Second Dimension: Model Types

This document suggests classifying YANG model types as either standard YANG models, vendor-specific YANG models, or vendor-specific extensions.

The suggested classification applies to both Network Element YANG Data Models and to Network Service YANG Data Models.

It is to be expected that real-world implementations of both Network Service and Network Element models will include a mix of all three types of models.

3.1. Standard YANG Models

Standard YANG models are published by standards defining organizations (SDOs). While there is no formal definition of what construes an SDO, a common feature is that they publish specifications along specific processes with content that reflects some sort of membership consensus. The specifications are developed for wide use among the membership or for audiences beyond that.

The lifecycle of these models are driven by the editing cycle of the specification and not tied to a specific implementation.

Examples of SDOs in the networking industry are the IETF, the IEEE and the MEF.

3.2. Vendor-specific YANG Models

Vendor-specific YANG models are developed by organizations with the intent to support a specific set of implementations under control of that organization. The intent of these models range from providing openly published YANG models that may eventually be contributed back to, or adopted by an SDO, to strictly internal YANG models not intended for external consumption.

The lifecycle of these models are generally aligned with the release cycle of the product or open source software project deliverables.

It is worth noting that there is an increasing amount of interaction between OSS projects and SDOs in the networking industry. This includes OSS projects implementing published standards as well as OSS

projects contributing content to standards defining organization processes.

3.3. Vendor-specific Extensions

Vendors develop Vendor-specific Extensions to standard models using YANG constructs for extending data definitions of previously published models. This is done using the 'augment' statement that allows locally defined data trees to be augmented into locations in externally defined data trees.

Vendors use this to extend standard data models to cover the full scope of features in implementations, which commonly is broader than what is covered by the standard model.

4. Security Considerations

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

5. IANA Considerations

This document requests no action by IANA.

6. Acknowledgements

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

7. Change log [RFC Editor: Please remove]

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

version 2: added definitions for config and service models clarified second dimension of model classification. fixed typos

version 3: restructure and partial rewrite of section 2.

version 4: Language fixes

8. References

8.1. Normative References

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

8.2. Informative References

- [I-D.ietf-netmod-acl-model]
Bogdanovic, D., Sreenivasa, K., Huang, L., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
draft-ietf-netmod-acl-model-03 (work in progress), June 2015.
- [I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, J., Bogdanovic, D., and K.
Koushik, "Yang Data Model for OSPF Protocol", draft-ietf-
ospf-yang-02 (work in progress), September 2015.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed.,
and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,
<<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
<<http://www.rfc-editor.org/info/rfc7223>>.
- [Writable-MIB-Module-IESG-Statement]
"Writable MIB Module IESG Statement",
<[https://www.ietf.org/iesg/statement/writable-mib-
module.html](https://www.ietf.org/iesg/statement/writable-mib-module.html)>.
- [YANG-Data-Model-for-L3VPN-service-delivery]
"YANG Data Model for L3VPN service delivery",
<<https://tools.ietf.org/id/draft-l3vpn-service-yang>>.

Authors' Addresses

Dean Bogdanovic

Email: ivandean@gmail.com

Benoit Claise
Cisco Systems, Inc.

Email: bclaise@cisco.com

Carl Moberg
Cisco Systems, Inc.

Email: camoberg@cisco.com

Internet Draft
<draft-chen-netmod-enterprise-yang-namespace-00.txt>
Category: Informational
Expires in 6 months

I. Chen
Ericsson

October 16, 2015

Grammar for Enterprise YANG Module Namespace
<draft-chen-netmod-enterprise-yang-namespace-00.txt>

Status of this Memo

Distribution of this memo is unlimited.

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 date.

Copyright Notice

Copyright (c) 2015 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 defines the grammar to create enterprise YANG module namespaces that are globally unique, as required by the YANG modeling language.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. YANG Module Namespace Requirements and Recommendations	3
3. Enterprise YANG Module Namespace Grammar	4
4. Usage Example	4
5. IANA Considerations	5
6. Security Considerations	5
7. References	5

1. Introduction

The use of a standard data modeling language YANG [RFC6020] together with Network Configuration Protocol (NETCONF) [RFC6241] allows for the creation of a standard network configuration interface. YANG further allows vendors to customize standard YANG modules and to create enterprise YANG modules that adapt standard YANG modules to different devices and features. To identify YANG modules, [RFC6020] requires YANG module namespaces of all YANG modules, both standard YANG modules and enterprise YANG modules, be globally unique. [RFC6020] also recommends that enterprise YANG modules have module names that are globally unique.

Based the module naming convention recommended in [RFC6020], this document defines the grammar to create YANG module namespaces for enterprise YANG modules that result in globally unique namespaces.

1.1. Terminology

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

2. YANG Module Name and Namespace Requirements and Recommendations

[RFC6020] consists of several YANG module name and namespace requirements and recommendations.

[RFC6020] Section 5.1 paragraph 3 states that

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.

[RFC6020] Section 5.3 paragraph 3 states that

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.

[RFC6020] Section 6.2.1 paragraph 1 states that

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.

The requirement in [RFC6020] Section 6.2.1 means that even enterprise YANG module names must be globally unique. The recommendation in [RFC6020] Section 5.1 means that it is desirable to define enterprise YANG modules names to use an organization's name as a prefix.

3. Enterprise YANG Module Namespace Grammar

This section defines the grammar to create globally unique enterprise YANG module namespaces. The grammar defines a namespace to be a Uniform Resource Name (URN) composed of an organization's reverse registered domain name, followed by optional sub-domain hierarchies, and ending with the module name with the organization's name as the prefix.

<namespace> = urn:<reverse-dns>:<sub-domain><module-name>

<reverse-dns> = An organization's registered domain name in reverse

<sub-domain> = Empty string or
 additional levels of hierarchy defined within a domain
 in which each level is delimited by colons

<module-name> = <organization-prefix>-<function>

<function> = A string that describes the function provided by the
 YANG module

4. Usage Example

Suppose a vendor has a registered domain name "example.com". This vendor has also chosen to place all of its YANG modules under the

"yang" sub-domain. Following the enterprise YANG module namespace grammar described in this document, the vendor ends up with the patterns below.

```
<reverse-dns> = com:example
<sub-domain> = yang
<namespace> = urn:com:example:yang:<module-name>
<module-name> = example-<function>
```

As a result, this vendor's OSPF YANG module has the namespace "urn:com:example:yang:example-ospf".

5. Security Considerations

TBD.

6. IANA Considerations

TBD. Is it necessary to include a reference to the top-level "com" registry RFC?

7. References

7.1. Normative References

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

7.2. Informative References

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

Author's Address

I. Chen
ing-wher.chen@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

G.Galimberti, Ed.
Cisco
R.Kunze, Ed.
Deutsche Telekom
K. Lam, Ed.
Alcatel-Lucent
D. Hiremagalur, Ed.
G. G.Grammel, Ed.
Juniper
L.Fang, Ed.
G.Ratterree, Ed.
Microsoft
October 19, 2015

A YANG model to manage the optical interface parameters for an external
transponder in a WDM network
draft-dharini-netmod-dwdm-if-yang-00

Abstract

This memo defines a Yang model that translates the SNMP mib module defined in draft-galikunze-ccamp-dwdm-if-snmp-mib for managing single channel optical interface parameters of DWDM applications. This model is to support the optical parameters specified in ITU-T G.698.2 [ITU.G698.2] and application identifiers specified in ITU-T G.874.1 [ITU.G874.1] . Note that G.874.1 encompasses vendor-specific codes, which if used would make the interface a single vendor IaDI and could still be managed.

The Yang model defined in this memo can be used for Optical Parameters monitoring and/or configuration of the endpoints of the multi-vendor IaDI optical link.

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 April 21, 2016.

Copyright Notice

Copyright (c) 2015 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. The Internet-Standard Management Framework	4
3. Conventions	4
4. Overview	4
4.1. Optical Parameters Description	5
4.1.1. Rs-Ss Configuration	6
4.1.2. Table of Application Codes	7
4.2. Use Cases	7
4.3. Optical Interface for external transponder in a WDM network	7
5. Structure of the Yang Module	8
6. Yang Module	8
7. Security Considerations	13
8. IANA Considerations	14
9. Acknowledgements	14
10. Contributors	14
11. References	15
11.1. Normative References	15
11.2. Informative References	17
Appendix A. Change Log	18
Appendix B. Open Issues	18
Authors' Addresses	18

1. Introduction

This memo defines a Yang model that translates the SNMP mib module defined in draft-galikunze-ccamp-dwdm-if-snmp-mib for managing single channel optical interface parameters of DWDM applications, using the approach specified in G.698.2. This model is to support the optical parameters specified in ITU-T G.698.2 [ITU.G698.2], application identifiers specified in ITU-T G.874.1 [ITU.G874.1] and the Optical Power at Transmitter and Receiver side. Note that G.874.1 encompasses vendor-specific codes, which if used would make the interface a single vendor IaDI and could still be managed.'

[Editor's note: In G.698.2 this corresponds to the optical path from point S to R; network media channel is also used and explained in draft-ietf-ccamp-flexi-grid-fwk-02]

Management will be performed at the edges of the network media channel (i.e., at 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 in G.7710/Y.1701 [ITU-T G.7710] and G.874.1 [ITU.G874.1].

G.698.2 [ITU.G698.2] is primarily intended for metro applications that 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 explicitly specify the details of the optical network between reference point Ss and Rs, e.g., the passive and active elements or details of the design. 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] and included in the Application Identifier defined in G.874.1 [ITU.G874.1] and G.872 [ITU.G872], plus a few optical parameters not included in the G.698.2 application code specification.

This draft refers and supports the draft-kdkgall-ccamp-dwdm-if-mng-ctrl-fwk

The building of a yang model describing the optical parameters defined in G.698.2 [ITU.G698.2], and reflected in G.874.1 [ITU.G874.1], allows the different vendors and operator to retrieve,

provision and exchange information across the G.698.2 multi-vendor IaDI in a standardized way. In addition to the parameters specified in ITU recommendations the Yang models support also the "vendor specific application identifier", the Tx and Rx power at the Ss and Rs points and the channel frequency.

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

Although RFC 3591 [RFC3591], which draft-galikunze-ccamp-DWDM-if-snmp-mib is extending, describes and defines the SNMP MIB of a number of key optical parameters, alarms and Performance Monitoring, as this RFC is over a decade old, it is primarily pre-OTN, and a more complete and up-to-date description of optical parameters and processes can be found in the relevant ITU-T Recommendations. The same considerations can be applied to the RFC 4054 [RFC4054].

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].

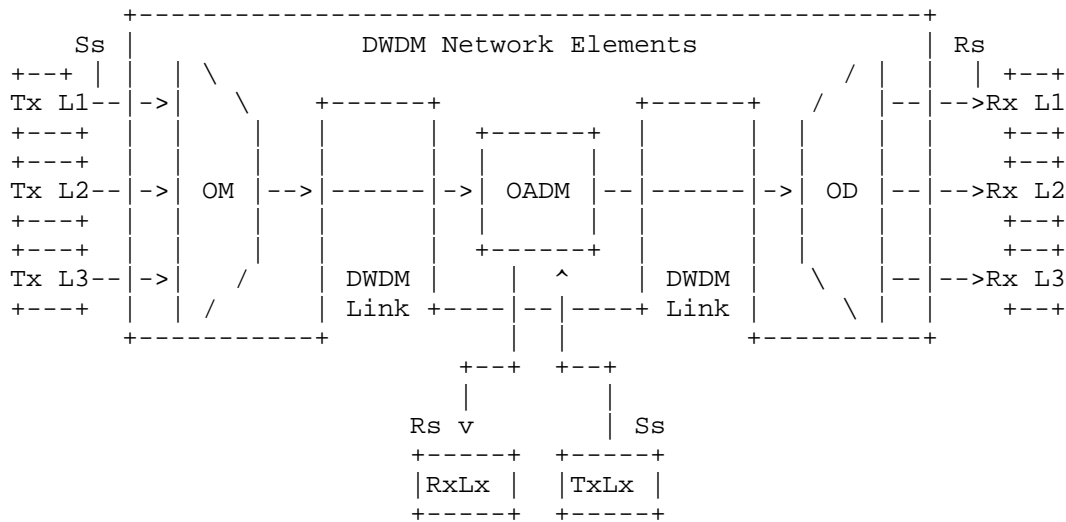
This memo specifies a Yang model for optical interfaces.

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 single-channel connection 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: External transponder in WDM networks

4.1. Optical Parameters Description

The link between the external transponders through a WDM network media channels are managed at the edges, i.e. at the transmitters (Tx) and receivers (Rx) attached to the S and R reference points respectively. The set of parameters that could be managed are defined by 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 (R) the parameter can be retrieve with a read, when (W) it can be provisioned by a write, (R,W) can be either read or written.

4.1.1.1. Rs-Ss Configuration

The Rs-Ss configuration table allows configuration of Central Frequency, 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).

Central frequency (see G.694.1 Table 1) (see G.694.1 Table 1):

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

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
Central frequency Value	G,S	G.694.1 S.6
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 Type:

This parameter indicates the transceiver type of application code at Ss and Rs as defined in [ITU.G874.1], that is used by this interface Standard = 0, PROPRIETARY = 1

The first 6 octets of the printable string will be the OUI (organizationally unique identifier) assigned to the vendor whose implementation generated the Application Identifier Code.

Application code Length:

The number of octets in the Application Code.

Application code:

This is the application code that is defined in G.698.2 or the vendor generated code which has the OUI.

4.2. Use Cases

The use cases are described in draft-kdkgall-ccamp-dwdm-if-mng-ctrl-fwk

4.3. Optical Interface for external transponder in a WDM network

The ietf-ext-xponder-wdm-if is an augment to the ietf-interface. It allows the user to set the application code/vendor transceiver class/Central frequency and the output power. The module can also be used to get the list of supported application codes/transceiver class and also the Central frequency/output power/input power of the interface.

```

module: ietf-ext-xponder-wdm-if
augment /if:interfaces/if:interface:
  +--rw optIfOChRsSs
    +--rw if-current-application-code
      |   +--rw application-code-id      uint8
      |   +--rw application-code-type   uint8
      |   +--rw application-code-length uint8
      |   +--rw application-code?      string
    +--ro if-supported-application-codes
      |   +--ro number-application-codes-supported? uint32
      |   +--ro application-codes-list* [application-code-id]
      |     +--ro application-code-id      uint8
      |     +--ro application-code-type   uint8
      |     +--ro application-code-length uint8
      |     +--ro application-code?      string
    +--rw output-power?                  int32
    +--ro input-power?                  int32
    +--rw central-frequency?            uint32

  notifications:
  +---n opt-if-och-central-frequency-change
    |   +--ro if-name?          leafref
    |   +--ro new-central-frequency
    |     +--ro central-frequency? uint32
  +---n opt-if-och-application-code-change
    |   +--ro if-name?          leafref
    |   +--ro new-application-code
    |     +--ro application-code-id? uint8
    |     +--rw application-code-type uint8
    |     +--rw application-code-length uint8
    |     +--ro application-code?   string

```

5. Structure of the Yang Module

ietf-ext-xponder-wdm-if is a top level model for the support of this feature.

6. Yang Module

The ietf-ext-xponder-wdm-if is defined as an extension to ietf interfaces.

<CODE BEGINS> file "ietf-ext-xponder-wdm-if.yang"

```
module ietf-ext-xponder-wdm-if {
  namespace "urn:ietf:params:xml:ns:yang:ietf-ext-xponder-wdm-if";
  prefix ietf-ext-xponder-wdm-if;

  import ietf-interfaces {
    prefix if;
  }

  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:dharithi@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 "2015-06-24" {
    description
      "Revision 4.0";

    reference
      " draft-dharini-netmod-dwdm-if-yang 3.0";
  }
  revision "2015-02-24" {
    description
```



```
        "Revision 3.0";

        reference
            " draft-dharini-netmod-dwdm-if-yang 3.0";
    }
    revision "2014-11-10" {
        description
            "Revision 2.0";
        reference
            " ";
    }
    revision "2014-10-14" {
        description
            "Revision 1.0";
        reference
            " ";
    }
    revision "2014-05-10" {
        description
            "Initial revision.";
        reference
            "RFC XXXX: A YANG Data Model for Optical
            Management of an Interface for an external
            transponder in a WDM network";
    }
}
```

```
grouping opt-if-och-application-code {
    description "Application code entity.";
    leaf application-code-id {
        type uint8 {
            range "1..255";
        }
        description
            "Id for the Application code";
    }
    leaf application-code-type {
        type uint8 {
            range "0..1";
        }
        description
            "Type for the Application code
            0 - Standard, 1 - Proprietary
            When the Type is Proprietary, then the
            first 6 octets of the application-code
```

```
        will be the OUI (organizationally unique
        identifier)";

    }
    leaf application-code-length {
        type uint8 {
            range "1..255";
        }
        description
            "Number of octets in the Application code";
    }
    leaf application-code {
        type string {
            length "1..255";
        }
        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 opt-if-och-application-code-list {
    description "List of Application codes group.";
    leaf number-application-codes-supported {
        type uint32;
        description "Number of Application codes
            supported by this interface";
    }
    list application-code-list {
        key "application-code-id";
        description "List of the application codes";
        uses opt-if-och-application-code;
    }
}

grouping opt-if-och-power {
    description "Interface optical Power";
    leaf output-power {
        type int32;
        units ".01dbm";
    }
}
```

```
        description "The output power for this interface in
                    .01 dBm.";
    }

    leaf input-power {
        type int32;
        units ".01dbm";
        config false;
        description "The current input power of this
                    interface";
    }
}

grouping opt-if-och-central-frequency {
    description "Interface Central Frequency";
    leaf central-frequency {
        type uint32;
        description "This parameter indicate This parameter
                    indicates the frequency of this interface ";
    }
}

notification opt-if-och-central-frequency-change {
    description "A change of Central Frequency has been
                detected.";
    leaf "if-name" {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        description "Interface name";
    }
    container opt-if-och-central-frequency {
        description "The new Central Frequency of the
                    interface";
        uses opt-if-och-central-frequency;
    }
}

notification opt-if-och-application-code-change {
    description "A change of Application code has been
                detected.";
    leaf "if-name" {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        description "Interface name";
    }
}
```

```
        container new-application-code {
            description "The new application code for the
                interface";
            uses opt-if-och-application-code;
        }
    }

    augment "/if:interfaces/if:interface" {
        description "Parameters for an optical interface";
        container optIfOChRsSs {
            description "RsSs path configuration for an interface";
            container if-current-application-code {
                description "Current Application code of the
                    interface";
                uses opt-if-och-application-code;
            }

            container if-supported-application-codes {
                config false;
                description "Supported Application codes of
                    the interface";
                uses opt-if-och-application-code-list;
            }

            uses opt-if-och-power;

            uses opt-if-och-central-frequency;
        }
    }
}
```

<CODE ENDS>

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-ext-xponder-wdm-if

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-ext-xponder-wdm-if reference: RFC XXXX

9. Acknowledgements

Gert Grammel is partly funded by European Union Seventh Framework Programme under grant agreement 318514 CONTENT.

10. Contributors

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

Bernd Zeuner
Deutsche Telekom
Darmstadt
Germany
email B.Zeuner@telekom.de

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

11. References

11.1. Normative References

- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <<http://www.rfc-editor.org/info/rfc2863>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, <<http://www.rfc-editor.org/info/rfc2579>>.
- [RFC2580] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Conformance Statements for SMIv2", STD 58, RFC 2580, DOI 10.17487/RFC2580, April 1999, <<http://www.rfc-editor.org/info/rfc2580>>.
- [RFC3591] Lam, H-K., Stewart, M., and A. Huynh, "Definitions of Managed Objects for the Optical Interface Type", RFC 3591, DOI 10.17487/RFC3591, September 2003, <<http://www.rfc-editor.org/info/rfc3591>>.
- [RFC6205] Otani, T., Ed. and D. Li, Ed., "Generalized Labels for Lambda-Switch-Capable (LSC) Label Switching Routers", RFC 6205, DOI 10.17487/RFC6205, March 2011, <<http://www.rfc-editor.org/info/rfc6205>>.
- [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.

11.2. Informative References

- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <<http://www.rfc-editor.org/info/rfc3410>>.

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<http://www.rfc-editor.org/info/rfc2629>>.
- [RFC4181] Heard, C., Ed., "Guidelines for Authors and Reviewers of MIB Documents", BCP 111, RFC 4181, DOI 10.17487/RFC4181, September 2005, <<http://www.rfc-editor.org/info/rfc4181>>.
- [I-D.kdkgall-ccamp-dwdm-if-mng-ctrl-fwk]
Kunze, R., Grammel, G., Beller, D., and G. Galimberti, "A framework for Management and Control of G.698.2 optical interface parameters", draft-kdkgall-ccamp-dwdm-if-mng-ctrl-fwk-00 (work in progress), October 2015.
- [RFC4054] Strand, J., Ed. and A. Chiu, Ed., "Impairments and Other Constraints on Optical Layer Routing", RFC 4054, DOI 10.17487/RFC4054, May 2005, <<http://www.rfc-editor.org/info/rfc4054>>.

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 Santa Maria Molgora, 48 c
20871 - Vimercate
Italy

Phone: +390392091462
Email: ggalimbe@cisco.com

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

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

Kam Lam (editor)
Alcatel-Lucent
USA

Phone: +1 732 331 3476
Email: kam.lam@alcatel-lucent.com

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

Email: dharinih@juniper.net

Gert Grammel (editor)
Juniper
Oskar-Schlemmer Str. 15
80807 Muenchen
Germany

Phone: +49 1725186386
Email: ggrammel@juniper.net

Luyuan Fang (editor)
Microsoft
5600 148th Ave NE
Redmond, WA 98502
USA

Email: lufang@microsoft.com

Gary Ratterree (editor)
Microsoft
5600 148th Ave NE
Redmond, WA 98502
USA

Email: gratt@microsoft.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
J. Dong
Huawei Technologies
D. Romascanu
Avaya
October 19, 2015

A YANG Data Model for Entity Managemet
draft-entitydt-netmod-entity-00

Abstract

This document defines a YANG data model for the management of multiple physical entities managed by a single server.

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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. Terminology	2
1.1.1. Tree Diagrams	2
2. Objectives	3
3. Entity Data Model	3
3.1. The Physical Entry Lists	5
4. Relationship to ENTITY-MIB	5
5. Relationship to ENTITY-SENSOR-MIB	6
6. Relationship to ENTITY-STATE-MIB	6
7. Entity YANG Module	6
8. IANA Considerations	34
9. Security Considerations	35
10. Acknowledgements	35
11. Normative References	35
Authors' Addresses	35

1. Introduction

This document defines a YANG data model for the management of multiple physical entities managed by a single server.

The data model includes configuration data and state data (status information and counters for the collection of statistics).

1.1. Terminology

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

1.1.1. 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 data (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. Objectives

This section describes some of the design objectives for the entity model.

- o There are many common properties used to identify the entities, which need to be supported in the entity data module.
- o There are many important information and states about the entities, which needs to be collected from the devices which support the entity data model.
- o The entity data model SHOULD be suitable for new implementations to use as is.
- o The entity data model defined in this document can be implemented on a system that also implements ENTITY-MIB, thus the mapping between the entity data model and ENTITY-MIB SHOULD be clear.

3. Entity Data Model

This document defines the YANG module "ietf-entity", which has the following structure:

```

module: ietf-entity
  +--ro entity-state
    |   +--ro last-change?          yang:date-and-time
    |   +--ro physical-entity* [name]
    |     |   +--ro name                string
    |     |   +--ro class?              identityref
    |     |   +--ro physical-index?     int32 {entity-mib}?
    |     |   +--ro description?        string
    |     |   +--ro contained-in*       -> ../../physical-entity/name
    |     |   +--ro contains-child*     -> ../../physical-entity/name
    |     |   +--ro parent-rel-pos?     int32
    |     |   +--ro hardware-rev?       string
    |     |   +--ro firmware-rev?      string
    |     |   +--ro software-rev?      string
    |     |   +--ro serial-num?        string

```

```

|      +--ro mfg-name?                string
|      +--ro model-name?              string
|      +--ro alias?                   string
|      +--ro asset-id?                string
|      +--ro is-fru?                  boolean
|      +--ro mfg-date?                yang:date-and-time
|      +--ro uri*                      inet:uri
|      +--ro uuid?                    yang:uuid
|      +--ro state {entity-state}?
|      |      +--ro state-last-changed? yang:date-and-time
|      |      +--ro admin-state?        entity-admin-state
|      |      +--ro oper-state?         entity-oper-state
|      |      +--ro usage-state?        entity-usage-state
|      |      +--ro alarm-status?       entity-alarm-status
|      |      +--ro standby-status?     entity-standby-status
|      +--ro sensor-data {entity-sensor}?
|      |      +--ro data-type?          entity-sensor-data-type
|      |      +--ro data-scale?         entity-sensor-data-scale
|      |      +--ro precision?          entity-sensor-precision
|      |      +--ro value?              entity-sensor-value
|      |      +--ro oper-status?        entity-sensor-status
|      |      +--ro sensor-units-display? string
|      |      +--ro value-timestamp?    yang:date-and-time
|      |      +--ro value-update-rate?  uint32
+--rw entity {entity-config}?
|   +--rw physical-entity* [name]
|   |   +--rw name                string
|   |   +--rw serial-num?         string
|   |   +--rw alias?              string
|   |   +--rw asset-id?           string
|   |   +--rw uri*                 inet:uri
|   |   +--rw admin-state?        entity-admin-state {entity-state}?

```

notifications:

```

+---n ent-config-change
+---n ent-state-oper-enabled {entity-state}?
|   +--ro name?                -> /entity-state/physical-entity
|                               /name
|   +--ro admin-state?         -> /entity-state/physical-entity
|                               /state/admin-state
|   +--ro alarm-status?        -> /entity-state/physical-entity
|                               /state/alarm-status
+---n ent-state-oper-disabled {entity-state}?
|   +--ro name?                -> /entity-state/physical-entity
|                               /name
|   +--ro admin-state?         -> /entity-state/physical-entity
|                               /state/admin-state
|   +--ro alarm-status?        -> /entity-state/physical-entity

```

/state/alarm-status

3.1. The Physical Entry Lists

The data model for physical entities presented in this document uses a flat list of entities. Each entity in the list is identified by its name. Furthermore, each entity has a mandatory "class" leaf.

The "iana-entity" module defines YANG identities for the hardware types in the IANA-maintained "IANA-ENTITY-MIB" registry.

The "class" leaf is a YANG identity that describes the type of the hardware. Vendors are encouraged to either directly use one of the common IANA-defined identities, or derive a more specific identity from one of them.

There is one optional list of configured physical entities ("/entity/physical-entity"), and a separate list for the operational state of all physical entities ("/entity-state/physical-entity").

4. Relationship to ENTITY-MIB

If the device implements the ENTITY-MIB [RFC6933], each entry in the /entity-state/physical-entity list is mapped to one EntPhysicalEntry. Objects that are writable in the MIB are mapped to nodes in the /entity/physical-entity list.

The "physical-index" leaf MUST contain the value of the corresponding entPhysicalEntry's entPhysicalIndex.

The "class" leaf is mapped to both entPhysicalClass and entPhysicalVendorType. If the value of the "class" leaf is an identity that is either derived from or is one of the identities in the "iana-entity" module, then entPhysicalClass contains the corresponding IANAPhysicalClass enumeration value. Otherwise, entPhysicalClass contains the IANAPhysicalClass value "other(1)". Vendors are encouraged to define an identity (derived from an identity in "iana-entity" if possible) for each enterprise-specific registration identifier used for entPhysicalVendorType, and use that identity for the "class" leaf.

The following tables list the YANG data nodes with corresponding objects in the ENTITY-MIB.

YANG data node in /entity-state /physical-entity	ENTITY-MIB object
name	entPhysicalName
class	entPhysicalClass
	entPhysicalVendorType
physical-index	entPhysicalIndex
description	entPhysicalDescr
contained-in	entPhysicalContainedIn
contains-child	entPhysicalChildIndex
parent-rel-pos	entPhysicalParentRelPos
hardware-rev	entPhysicalHardwareRev
firmware-rev	entPhysicalFirmwareRev
software-rev	entPhysicalSoftwareRev
serial-num	entPhysicalSerialNum
mfg-name	entPhysicalMfgName
model-name	entPhysicalModelName
alias	entPhysicalAlias
asset-id	entPhysicalAssetID
is-fru	entPhysicalIsFRU
mfg-date	entPhysicalMfgDate
uri	entPhysicalUris
uuid	entPhysicalUUID

YANG data nodes and related ENTITY-MIB objects

5. Relationship to ENTITY-SENSOR-MIB

TBD relationship to [RFC3433].

6. Relationship to ENTITY-STATE-MIB

TBD relationship to [RFC4268].

7. Entity YANG Module

<CODE BEGINS> file "ietf-entity@2015-10-19.yang"

```

module ietf-entity {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-entity";
  prefix ent;

  import ietf-inet-types {
    prefix inet;
  }

```

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

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

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

  WG Chair: Thomas Nadeau
             <mailto:tnadeau@lucidvision.com>

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

  WG Chair: Kent Watsen
             <mailto:kwatsen@juniper.net>

  Editor:    Andy Bierman
             <mailto:andy@yumaworks.com>

  Editor:    Martin Bjorklund
             <mailto:mbj@tail-f.com>

  Editor:    ";

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

description
  "This module contains a collection of YANG definitions for
  managing physical entities.

  Copyright (c) 2015 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.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-10-19 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Entity Managemet";
}

/*
 * Features
 */

feature entity-mib {
  description
    "This feature indicates that the device implements
    the ENTITY-MIB.";
  reference "RFC 6933: Entity MIB (Version 4)";
}

feature entity-config {
  description
    "Indicates that the server supports configuration of
    physical entities.";
}

feature entity-state {
  description
    "Indicates the ENTITY-STATE-MIB objects are supported";
  reference "RFC 4268: Entity State MIB";
}

feature entity-sensor {
  description
    "Indicates the ENTITY-SENSOR-MIB objects are supported";
  reference "RFC 3433: Entity Sensor MIB";
}

/*
 * Typedefs
 */

typedef entity-admin-state {
  type enumeration {
    enum unknown {
      value 1;
      description
        "The resource is unable to report administrative state.";
    }
  }
}
```

```
    }
    enum locked {
        value 2;
        description
            "The resource is administratively prohibited from use.";
    }
    enum shutting-down {
        value 3;
        description
            "The resource usage is administratively limited to current
            instances of use.";
    }
    enum unlocked {
        value 4;
        description
            "The resource is not administratively prohibited from use.";
    }
}
description
    "Represents the various possible administrative states.";
reference "RFC 4268: EntityAdminState";
}

typedef entity-oper-state {
    type enumeration {
        enum unknown {
            value 1;
            description
                "The resource is unable to report operational state.";
        }
        enum disabled {
            value 2;
            description
                "The resource is totally inoperable.";
        }
        enum enabled {
            value 3;
            description
                "The resource is partially or fully operable.";
        }
        enum testing {
            value 4;
            description
                "The resource is currently being tested and cannot
                therefore report whether it is operational or not.";
        }
    }
}
description
```

```
    "Represents the possible values of operational states.";
    reference "RFC 4268: EntityOperState";
}

typedef entity-usage-state {
    type enumeration {
        enum unknown {
            value 1;
            description
                "The resource is unable to report usage state.";
        }
        enum idle {
            value 2;
            description
                "The resource is servicing no users.";
        }
        enum active {
            value 3;
            description
                "The resource is currently in use and it has sufficient
                spare capacity to provide for additional users.";
        }
        enum busy {
            value 4;
            description
                "The resource is currently in use, but it currently has
                no spare capacity to provide for additional users.";
        }
    }
    description
        "Represents the possible values of usage states.";
    reference "RFC 4268, EntityUsageState";
}

typedef entity-alarm-status {
    type bits {
        bit unknown {
            position 0;
            description
                "The resource is unable to report alarm state.";
        }
        bit under-repair {
            position 1;
            description
                "The resource is currently being repaired, which, depending
                on the implementation, may make the other values in this
                bit string not meaningful.";
        }
    }
}
```

```
    bit critical {
        position 2;
        description
            "One or more critical alarms are active against the
             resource.";
    }
    bit major {
        position 3;
        description
            "One or more major alarms are active against the resource.";
    }
    bit minor {
        position 4;
        description
            "One or more minor alarms are active against the resource.";
    }
    bit warning {
        position 5;
        description
            "One or more warning alarms are active against the resource.
             This alarm status is not defined in X.733.";
    }
    bit indeterminate {
        position 6;
        description
            "One or more alarms of whose perceived severity cannot be
             determined are active against this resource.
             This alarm status is not defined in X.733.";
    }
}
description
    "Represents the possible values of alarm status.
     An Alarm [RFC3877] is a persistent indication of an error or
     warning condition.

     When no bits of this attribute are set, then no active
     alarms are known against this entity and it is not under
     repair.";
reference "RFC 4268: EntityAlarmStatus";
}

typedef entity-standby-status {
    type enumeration {
        enum unknown {
            value 1;
            description
                "The resource is unable to report standby state.";
        }
    }
}
```

```
enum hot-standby {
  value 2;
  description
    "The resource is not providing service, but it will be
    immediately able to take over the role of the resource
    to be backed up, without the need for initialization
    activity, and will contain the same information as the
    resource to be backed up.";
}
enum cold-standby {
  value 3;
  description
    "The resource is to back up another resource, but will not
    be immediately able to take over the role of a resource
    to be backed up, and will require some initialization
    activity.";
}
enum providing-service {
  value 4;
  description
    "The resource is providing service.";
}
}
description
  "Represents the possible values of standby status.";
reference "RFC 4268: EntityStandbyStatus";
}

typedef entity-sensor-data-type {
  type enumeration {
    enum other {
      value 1;
      description
        "A measure other than those listed below.";
    }
    enum unknown {
      value 2;
      description
        "An unknown measurement, or arbitrary, relative numbers";
    }
    enum volts-AC {
      value 3;
      description
        "A measure of electric potential (alternating current).";
    }
    enum volts-DC {
      value 4;
      description
```

```
        "A measure of electric potential (direct current).";
    }
    enum amperes {
        value 5;
        description
            "A measure of electric current.";
    }
    enum watts {
        value 6;
        description
            "A measure of power.";
    }
    enum hertz {
        value 7;
        description
            "A measure of frequency.";
    }
    enum celsius {
        value 8;
        description
            "A measure of temperature.";
    }
    enum percent-RH {
        value 9;
        description
            "A measure of percent relative humidity.";
    }
    enum rpm {
        value 10;
        description
            "A measure of shaft revolutions per minute.";
    }
    enum cmm {
        value 11;
        description
            "A measure of cubic meters per minute (airflow).";
    }
    enum truth-value {
        value 12;
        description
            "Value is one of 1 (true) or 2 (false)";
    }
}
description
    "An node using this data type represents the Entity Sensor
    measurement data type associated with a physical sensor
    value. The actual data units are determined by examining an
    node of this type together with the associated
```


entity-sensor-data-scale node.

An node of this type SHOULD be defined together with nodes of type entity-sensor-data-scale and entity-sensor-precision. These three types are used to identify the semantics of an node of type entity-sensor-value.";
reference "RFC 3433: EntitySensorDataType";
}

```
typedef entity-sensor-data-scale {  
  type enumeration {  
    enum yocto {  
      value 1;  
      description  
        "Data scaling factor of 10^-24.";  
    }  
    enum zepto {  
      value 2;  
      description  
        "Data scaling factor of 10^-21.";  
    }  
    enum atto {  
      value 3;  
      description  
        "Data scaling factor of 10^-18.";  
    }  
    enum femto {  
      value 4;  
      description  
        "Data scaling factor of 10^-15.";  
    }  
    enum pico {  
      value 5;  
      description  
        "Data scaling factor of 10^-12.";  
    }  
    enum nano {  
      value 6;  
      description  
        "Data scaling factor of 10^-9.";  
    }  
    enum micro {  
      value 7;  
      description  
        "Data scaling factor of 10^-6.";  
    }  
    enum milli {  
      value 8;
```

```
    description
      "Data scaling factor of 10^-3.";
  }
  enum units {
    value 9;
    description
      "Data scaling factor of 10^0.";
  }
  enum kilo {
    value 10;
    description
      "Data scaling factor of 10^3.";
  }
  enum mega {
    value 11;
    description
      "Data scaling factor of 10^6.";
  }
  enum giga {
    value 12;
    description
      "Data scaling factor of 10^9.";
  }
  enum tera {
    value 13;
    description
      "Data scaling factor of 10^12.";
  }
  enum exa {
    value 14;
    description
      "Data scaling factor of 10^15.";
  }
  enum peta {
    value 15;
    description
      "Data scaling factor of 10^18.";
  }
  enum zetta {
    value 16;
    description
      "Data scaling factor of 10^21.";
  }
  enum yotta {
    value 17;
    description
      "Data scaling factor of 10^24.";
  }
}
```

```
}
description
  "An node using this data type represents a data scaling
  factor, represented with an International System of Units (SI)
  prefix. The actual data units are determined by examining an
  node of this type together with the associated
  entity-sensor-data-type.

  An node of this type SHOULD be defined together with nodes
  of type entity-sensor-data-type and entity-sensor-precision.
  Together, associated nodes of these three types are used to
  identify the semantics of an node of type
  entity-sensor-value.";
reference "RFC 3433: EntitySensorDataScale";
}

typedef entity-sensor-precision {
  type int32 {
    range "-8 .. 9";
  }
  description
    "An node using this data type represents a sensor
    precision range.

    An node of this type SHOULD be defined together with nodes
    of type entity-sensor-data-type and entity-sensor-data-scale.
    Together, associated nodes of these three types are used to
    identify the semantics of an node of type
    entity-sensor-value.

    If an node of this type contains a value in the range 1 to 9,
    it represents the number of decimal places in the fractional
    part of an associated entity-sensor-value fixed- point number.

    If an node of this type contains a value in the range -8 to
    -1, it represents the number of accurate digits in the
    associated entity-sensor-value fixed-point number.

    The value zero indicates the associated entity-sensor-value
    node is not a fixed-point number.

    Server implementors must choose a value for the associated
    entity-sensor-precision node so that the precision and
    accuracy of the associated entity-sensor-value node is
    correctly indicated.

    For example, a physical entity representing a temperature
    sensor that can measure 0 degrees to 100 degrees C in 0.1
```

```
    degree increments, +/- 0.05 degrees, would have an
    entity-sensor-precision value of '1', an
    entity-sensor-data-scale value of 'units', and an
    entity-sensor-value ranging from '0' to '1000'. The
    entity-sensor-value would be interpreted as 'degrees C * 10'.
    reference "RFC 3433: EntitySensorPrecision";
}
```

```
typedef entity-sensor-value {
  type int32 {
    range "-10000000000 .. 10000000000";
  }
  description
```

"An node using this data type represents an Entity Sensor value.

An node of this type SHOULD be defined together with nodes of type entity-sensor-data-type, entity-sensor-data-scale, and entity-sensor-precision. Together, associated nodes of those three types are used to identify the semantics of an node of this data type.

The semantics of an node using this data type are determined by the value of the associated entity-sensor-data-type node.

If the associated entity-sensor-data-type node is equal to 'voltsAC', 'voltsDC', 'amperes', 'watts', 'hertz', 'celsius', or 'cmm', then an node of this type MUST contain a fixed point number ranging from -999,999,999 to +999,999,999. The value -10000000000 indicates an underflow error. The value +10000000000 indicates an overflow error. The entity-sensor-precision indicates how many fractional digits are represented in the associated entity-sensor-value node.

If the associated entity-sensor-data-type node is equal to 'percentRH', then an node of this type MUST contain a number ranging from 0 to 100.

If the associated entity-sensor-data-type node is equal to 'rpm', then an node of this type MUST contain a number ranging from -999,999,999 to +999,999,999.

If the associated entity-sensor-data-type node is equal to 'truthvalue', then an node of this type MUST contain either the value 1 (true) or the value 2 (false).

If the associated entity-sensor-data-type node is equal to 'other' or 'unknown', then an node of this type MUST

```
        contain a number ranging from -10000000000 to 10000000000.";
        reference "RFC 3433: EntitySensorValue";
    }

typedef entity-sensor-status {
    type enumeration {
        enum ok {
            value 1;
            description
                "Indicates that the server can obtain the sensor value.";
        }
        enum unavailable {
            value 2;
            description
                "Indicates that the server presently cannot obtain the
                 sensor value.";
        }
        enum nonoperational {
            value 3;
            description
                "Indicates that the server believes the sensor is broken.
                 The sensor could have a hard failure (disconnected wire),
                 or a soft failure such as out-of-range, jittery, or wildly
                 fluctuating readings.";
        }
    }
    description
        "An node using this data type represents the operational
         status of a physical sensor.";
    reference "RFC 3433: EntitySensorStatus";
}

/*
 * Identities
 */

identity entity-physical-class {
    description
        "This identity is the base for all physical entity class
         identifiers.";
}

/*
 * Operational state data nodes
 */

container entity-state {
```

```
config false;
description
  "Data nodes for the operational state of physical entities.";

leaf last-change {
  type yang:date-and-time;
  description
    "The time the '/entity-state/physical-entity' list changed.";
}

list physical-entity {
  key name;
  description
    "List of physical entities";
  reference "RFC 6933: entPhysicalEntry";

  leaf name {
    type string;
    description
      "Administrative name assigned to this physical entity.
      No restrictions apply. Not required to be the same as
      entPhysicalName.";
  }

  leaf class {
    type identityref {
      base entity-physical-class;
    }
    mandatory true;
    description
      "An indication of the general hardware type
      of the physical entity.";
    reference "RFC 6933: entPhysicalClass";
  }

  leaf physical-index {
    if-feature entity-mib;
    type int32 {
      range "1..2147483647";
    }
    description
      "The entPhysicalIndex for the entPhysicalEntry represented
      by this list entry.";
    reference "RFC 6933: entPhysicalIndex";
  }

  leaf description {
    type string;
  }
}
```

```
    description
      "A textual description of physical entity. This node
       should contain a string that identifies the manufacturer's
       name for the physical entity and should be set to a
       distinct value for each version or model of the physical
       entity.";
    reference "RFC 6933: entPhysicalDescr";
  }

  leaf-list contained-in {
    type leafref {
      path "../../physical-entity/name";
    }
    description
      "The name of the physical entity that 'contains'
       this physical entity.";
    reference "RFC 6933: entPhysicalContainedIn";
  }

  leaf-list contains-child {
    type leafref {
      path "../../physical-entity/name";
    }
    description
      "The name of the contained physical entity.";
    reference "RFC 6933: entPhysicalChildIndex";
  }

  leaf parent-rel-pos {
    type int32 {
      range "0 .. 2147483647";
    }
    description
      "An indication of the relative position of this child
       component among all its sibling components. Sibling
       components are defined as physical entities that share the
       same instance values of each of the contained-in
       and class elements.";
    reference "RFC 6933: entPhysicalParentRelPos";
  }

  leaf hardware-rev {
    type string;
    description
      "The vendor-specific hardware revision string for the
       physical entity. The preferred value is the hardware
       revision identifier actually printed on the component
       itself (if present).";
```

```
    reference "RFC 6933: entPhysicalHardwareRev";
  }

  leaf firmware-rev {
    type string;
    description
      "The vendor-specific firmware revision string for the
       physical entity.";
    reference "RFC 6933: entPhysicalFirmwareRev";
  }

  leaf software-rev {
    type string;
    description
      "The vendor-specific software revision string for the
       physical entity.";
    reference "RFC 6933: entPhysicalSoftwareRev";
  }

  leaf serial-num {
    type string;
    description
      "The vendor-specific serial number string for the physical
       entity. The preferred value is the serial number string
       actually printed on the component itself (if present).

       If a serial number has been configured for this entity in
       /entity/physical-entity/serial-num, this node contains the
       configured value.";
    reference "RFC 6933: entPhysicalSerialNum";
  }

  leaf mfg-name {
    type string;
    description
      "The name of the manufacturer of this physical component.
       The preferred value is the manufacturer name string
       actually printed on the component itself (if present).

       Note that comparisons between instances of the model-name,
       firmware-rev, software-rev, and the serial-num nodes are
       only meaningful amongst physical entities with the same
       value of mfg-name.

       If the manufacturer name string associated with the
       physical component is unknown to the server, then this
       node will contain a zero-length string.";
    reference "RFC 6933: entPhysicalMfgName";
  }
```



```
}

leaf model-name {
  type string;
  description
    "The vendor-specific model name identifier string associated
    with this physical component.  The preferred value is the
    customer-visible part number, which may be printed on the
    component itself.

    If the model name string associated with the physical
    component is unknown to the server, then this node will
    contain a zero-length string.";
  reference "RFC 6933: entPhysicalModelName";
}

leaf alias {
  type string {
    length "0 .. 32";
  }
  description
    "An 'alias' name for the physical entity, as specified by
    a network manager, and provides a non-volatile 'handle'
    for the physical entity.

    If an alias has been configured for this entity in
    /entity/physical-entity/alias, this node contains the
    configured value.  If no such alias has been configured,
    the server may set the value of this node to a locally
    unique value.";
  reference "RFC 6933: entPhysicalAlias";
}

leaf asset-id {
  type string {
    length "0 .. 32";
  }
  description
    "This node is a user-assigned asset tracking identifier
    (as specified by a network manager) for the physical entity
    and provides non-volatile storage of this information.

    If an asset tracking identifier has been configured for
    this entity in /entity/physical-entity/asset-id, this node
    contains the configured value.";
  reference "RFC 6933: entPhysicalAssetID";
}
```

```
leaf is-fru {
  type boolean;
  description
    "This node indicates whether or not this physical entity
    is considered a 'field replaceable unit' by the vendor.  If
    this node contains the value 'true', then this
    physical entity identifies a field replaceable unit.  For
    all physical entities that represent components permanently
    contained within a field replaceable unit, the value
    'false' should be returned for this node.";
  reference "RFC 6933: entPhysicalIsFRU";
}

leaf mfg-date {
  type yang:date-and-time;
  description
    "The date of manufacturing of the managed entity.";
  reference "RFC 6933: entPhysicalMfgDate";
}

leaf-list uri {
  type inet:uri;
  description
    "This node contains identification information about the
    physical entity.

    If uris have been configured for this entity in
    /entity/physical-entity/uri, this node contains the
    configured values.";
  reference "RFC 6933: entPhysicalUris";
}

leaf uuid {
  type yang:uuid;
  description
    "A Universally Unique Identifier of the physical entity.";
  reference "RFC 6933: entPhysicalUUID";
}

container state {
  if-feature entity-state;
  description
    "State-related nodes";
  reference "RFC 4268: Entity State MIB";

  leaf state-last-changed {
    type yang:date-and-time;
    description
```

"The date and time when the value of any of the admin-state, oper-state, usage-state, alarm-status, or standby-status changed for this entity.

If there has been no change since the last re-initialization of the local system, this node contains the date and time of local system initialization. If there has been no change since the entity was added to the local system, this node contains the date and time of the insertion.";

reference "RFC 4268: entStateLastChanged";

}

leaf admin-state {
 type entity-admin-state;
 description

 "The administrative state for this entity.

This node refers to an entities administrative permission to service both other entities within its containment hierarchy as well other users of its services defined by means outside the scope of this module.

Some physical entities exhibit only a subset of the remaining administrative state values. Some entities cannot be locked, and hence this node exhibits only the 'unlocked' state. Other entities cannot be shutdown gracefully, and hence this node does not exhibit the 'shutting-down' state.";

reference "RFC 4268: entStateAdmin";

}

leaf oper-state {
 type entity-oper-state;
 description

 "The operational state for this entity.

Note that this node does not follow the administrative state. An administrative state of down does not predict an operational state of disabled.

Note that some implementations may not be able to accurately report oper-state while the admin-state node has a value other than 'unlocked'. In these cases, this node MUST have a value of 'unknown'.";

reference "RFC 4268: entStateOper";

}

```
leaf usage-state {
  type entity-usage-state;
  description
    "The usage state for this entity.

    This node refers to an entity's ability to service more
    physical entities in a containment hierarchy.

    Some entities will exhibit only a subset of the usage state
    values. Entities that are unable to ever service any
    entities within a containment hierarchy will always have a
    usage state of 'busy'. Some entities will only ever be
    able to support one entity within its containment hierarchy
    and will therefore only exhibit values of 'idle' and
    'busy'.";
  reference "RFC 4268, entStateUsage";
}

leaf alarm-status {
  type entity-alarm-status;
  description
    "The alarm status for this entity. It does not include
    the alarms raised on child components within its
    containment hierarchy.";
  reference "RFC 4268: entStateAlarm";
}

leaf standby-status {
  type entity-standby-status;
  description
    "The standby status for this entity.

    Some entities will exhibit only a subset of the
    remaining standby state values. If this entity
    cannot operate in a standby role, the value of this
    node will always be 'providing-service'.";
  reference "RFC 4268: entStateStandby";
}

container sensor-data {
  when 'derived-from-or-self(..../class,
                                "iana-entity", "sensor")' {
    description
      "Sensor data nodes present for any entity of type 'sensor'";
  }
  if-feature entity-sensor;
  description
```

```
"Sensor-related nodes.";
reference "RFC 3433: Entity Sensor MIB";

leaf data-type {
  type entity-sensor-data-type;
  description
    "The type of data units associated with the
     sensor value";
  reference "RFC 3433: entPhySensorType";
}

leaf data-scale {
  type entity-sensor-data-scale;
  description
    "The (power of 10) scaling factor associated
     with the sensor value";
  reference "RFC 3433: entPhySensorScale";
}

leaf precision {
  type entity-sensor-precision;
  description
    "The number of decimal places of precision
     associated with the sensor value";
  reference "RFC 3433: entPhySensorPrecision";
}

leaf value {
  type entity-sensor-value;
  description
    "The most recent measurement obtained by the server
     for this sensor.";
  reference "RFC 3433: entPhySensorValue";
}

leaf oper-status {
  type entity-sensor-status;
  description
    "The operational status of the sensor.";
  reference "RFC 3433: entPhySensorOperStatus";
}

leaf sensor-units-display {
  type string;
  description
    "A textual description of the data units that should be
     used in the display of the sensor value.";
  reference "RFC 3433: entPhySensorUnitsDisplay";
}
```

```
    }

    leaf value-timestamp {
      type yang:date-and-time;
      description
        "The time the status and/or value of this sensor was
        last obtained by the server.";
      reference "RFC 3433: entPhySensorValueTimeStamp";
    }

    leaf value-update-rate {
      type uint32;
      units "milliseconds";
      description
        "An indication of the frequency that the server updates
        the associated 'value' node, representing in
        milliseconds. The value zero indicates:

        - the sensor value is updated on demand (e.g.,
          when polled by the server for a get-request),
        - the sensor value is updated when the sensor
          value changes (event-driven),
        - the server does not know the update rate.";
      reference "RFC 3433: entPhySensorValueUpdateRate";
    }
  }
}

/*
 * Configuration data nodes
 */

container entity {
  if-feature entity-config;
  description
    "Configuration parameters for physical entities.";

  list physical-entity {
    key name;
    description
      "List of configuration data for physical entities.";

    leaf name {
      type string;
      description
        "Administrative name assigned to this physical entity.
        No restrictions apply.";
    }
  }
}
```

```
}

leaf serial-num {
  type string;
  description
    "The vendor-specific serial number string for the physical
    entity. The preferred value is the serial number string
    actually printed on the component itself (if present).

    This node is indented to be used for physical entities
    for which the server cannot determine the serial number.";
  reference "RFC 6933: entPhysicalSerialNum";
}

leaf alias {
  type string {
    length "0 .. 32";
  }
  description
    "This node is an 'alias' name for the physical entity, as
    specified by a network manager, and provides a non-volatile
    'handle' for the physical entity.";
  reference "RFC 6933: entPhysicalAlias";
}

leaf asset-id {
  type string {
    length "0 .. 32";
  }
  description
    "This node is a user-assigned asset tracking identifier
    (as specified by a network manager) for the physical entity";
  reference "RFC 6933: entPhysicalAssetID";
}

leaf-list uri {
  type inet:uri;
  description
    "This node contains identification information about the
    physical entity.";
  reference "RFC 6933: entPhysicalUris";
}

leaf admin-state {
  if-feature entity-state;
  type entity-admin-state;
  description
    "The administrative state for this entity.
```

This node refers to an entity's administrative permission to service both other entities within its containment hierarchy as well other users of its services defined by means outside the scope of this module.

Some physical entities exhibit only a subset of the remaining administrative state values. Some entities cannot be locked, and hence this node exhibits only the 'unlocked' state. Other entities cannot be shutdown gracefully, and hence this node does not exhibit the 'shutting-down' state.";

```
reference "RFC 4268, entStateAdmin";
    }
  }
}

/*
 * Notifications
 */

notification ent-config-change {
  description
    "An ent-config-change notification is generated when the value
    of /entity-state/last-change changes.";
  reference "RFC 6933, entConfigChange";
}

notification ent-state-oper-enabled {
  if-feature entity-state;
  description
    "An ent-state-oper-enabled notification signifies that
    an entity has transitioned into the 'enabled' state.";

  leaf name {
    type leafref {
      path "/entity-state/physical-entity/name";
    }
    description
      "The name of the entity that has transitioned into the
      'enabled' state.";
  }
  leaf admin-state {
    type leafref {
      path "/entity-state/physical-entity/state/admin-state";
    }
    description
      "The administrative state for the entity.";
```



```
    }
    leaf alarm-status {
      type leafref {
        path "/entity-state/physical-entity/state/alarm-status";
      }
      description
        "The alarm status for the entity.";
    }
    reference "RFC 4268, entStateOperEnabled";
  }

notification ent-state-oper-disabled {
  if-feature entity-state;
  description
    "An ent-state-oper-disabled notification signifies that
     an entity has transitioned into the 'disabled' state.";

  leaf name {
    type leafref {
      path "/entity-state/physical-entity/name";
    }
    description
      "The name of the entity that has transitioned into the
       'disabled' state.";
  }
  leaf admin-state {
    type leafref {
      path "/entity-state/physical-entity/state/admin-state";
    }
    description
      "The administrative state for the entity.";
  }
  leaf alarm-status {
    type leafref {
      path "/entity-state/physical-entity/state/alarm-status";
    }
    description
      "The alarm status for the entity.";
  }
  reference "RFC 4268, entStateOperDisabled";
}
}
```

<CODE ENDS>

Move this to a separate document?:

```
<CODE BEGINS> file "iana-entity@2015-10-19.yang"

module iana-entity {
  namespace "urn:ietf:params:xml:ns:yang:iana-entity";
  prefix ianaent;

  import ietf-entity {
    prefix ent;
  }

  organization "IANA";
  contact
    "      Internet Assigned Numbers Authority

    Postal: ICANN
           4676 Admiralty Way, Suite 330
           Marina del Rey, CA 90292

    Tel:    +1 310 823 9358
    <mailto:iana@iana.org>";

  description
    "IANA defined identities for physical class.";
  reference
    "https://www.iana.org/assignments/ianaentity-mib/ianaentity-mib";

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

  // RFC Ed.: update the date below with the date of RFC publication
  // and remove this note.
  revision 2015-10-19 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A YANG Data Model for Entity Managemet";
  }

  identity unknown {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
      is unknown to the server.";
  }

  identity chassis {
    base ent:entity-physical-class;
```

```
    description
      "This identity is applicable if the physical entity class
       is an overall container for networking equipment.  Any class of
       physical entity, except a stack, may be contained within a
       chassis; a chassis may only be contained within a stack.";
  }

  identity backplane {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is some sort of device for aggregating and forwarding
       networking traffic, such as a shared backplane in a modular
       ethernet switch.  Note that an implementation may model a
       backplane as a single physical entity, which is actually
       implemented as multiple discrete physical components (within a
       chassis or stack).";
  }

  identity container {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is capable of containing one or more removable physical
       entities, possibly of different types.  For example, each
       (empty or full) slot in a chassis will be modeled as a
       container.  Note that all removable physical entities should be
       modeled within a container entity, such as field-replaceable
       modules, fans, or power supplies.  Note that all known
       containers should be modeled by the agent, including empty
       containers.";
  }

  identity power-supply {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is a power-supplying component.";
  }

  identity fan {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is a fan or other heat-reduction component.";
  }

  identity sensor {
```

```
base ent:entity-physical-class;
description
  "This identity is applicable if the physical entity class
   is some sort of sensor, such as a temperature sensor within a
   router chassis.";
}

identity module {
  base ent:entity-physical-class;
  description
    "This identity is applicable if the physical entity class
     is some sort of self-contained sub-system. If a 'module'
     entity is removable, then it should be modeled within a
     container entity; otherwise, it should be modeled directly
     within another physical entity (e.g., a chassis or another
     module).";
}

identity port {
  base ent:entity-physical-class;
  description
    "This identity is applicable if the physical entity class
     is some sort of networking port, capable of receiving and/or
     transmitting networking traffic.";
}

identity stack {
  base ent:entity-physical-class;
  description
    "This identity is applicable if the physical entity class
     is some sort of super-container (possibly virtual) intended to
     group together multiple chassis entities. A stack may be
     realized by a 'virtual' cable, a real interconnect cable
     attached to multiple chassis, or multiple interconnect cables.
     A stack should not be modeled within any other physical
     entities, but a stack may be contained within another stack.
     Only chassis entities should be contained within a stack.";
}

identity cpu {
  base ent:entity-physical-class;
  description
    "This identity is applicable if the physical entity class
     is some sort of central processing unit.";
}

identity energy-object {
  base ent:entity-physical-class;
```

```
    description
      "This identity is applicable if the physical entity class
       is some sort of energy object, i.e., a piece of equipment that
       is part of or attached to a communications network that is
       monitored, controlled, or aids in the management of another
       device for Energy Management.";
  }

  identity battery {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is some sort of battery.";
  }

  identity storage-drive {
    base ent:entity-physical-class;
    description
      "This identity is applicable if the physical entity class
       is some sort of entity with data storage capability as main
       functionality, e.g., disk drive (HDD), solid state device
       (SSD), hybrid (SSHD), object storage (OSD) or other.";
  }
}
```

<CODE ENDS>

8. IANA Considerations

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

URI: TBD

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-entity
namespace: TBD
prefix:    ent
reference: RFC XXXX
```

9. Security Considerations

TBD

10. Acknowledgements

TBD

11. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3433] Bierman, A., Romascanu, D., and K. Norseth, "Entity Sensor Management Information Base", RFC 3433, DOI 10.17487/RFC3433, December 2002, <<http://www.rfc-editor.org/info/rfc3433>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC4268] Chisholm, S. and D. Perkins, "Entity State MIB", RFC 4268, DOI 10.17487/RFC4268, November 2005, <<http://www.rfc-editor.org/info/rfc4268>>.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <<http://www.rfc-editor.org/info/rfc6933>>.

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Jie Dong
Huawei Technologies

Email: jie.dong@huawei.com

Dan Romascanu
Avaya

Email: dromasca@avaya.com

NETMOD WG
Internet-Draft
Intended status: Informational
Expires: April 18, 2016

I. Farrer
Q. Sun
S. Zoric
Deutsche Telekom AG
M. Abrahamsson
T-Systems
October 16, 2015

YANG Models Required for Managing Customer Premises Equipment (CPE)
Devices
draft-faq-netmod-cpe-yang-profile-00

Abstract

This document collects together the YANG models necessary for managing NETCONF-enabled Customer Premises Equipment (CPE) devices.

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 18, 2016.

Copyright Notice

Copyright (c) 2015 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. Terminology	3
3. Management Requirements	3
3.1. Interfaces	3
3.1.1. Requirements	4
3.1.2. Development Status of Relevant YANG Models	4
3.2. IP Management	5
3.2.1. Requirements	5
3.2.2. Development of Relevant YANG Models	5
3.3. Routing and Multicast Management	5
3.3.1. Requirements	5
3.3.2. Development of Relevant YANG Models	6
3.4. CPE NETCONF Server Management	6
3.4.1. Requirements	6
3.4.2. Development Status of Relevant YANG Models	6
3.5. DHCP/SLAAC/ND Management	7
3.5.1. Requirements	7
3.5.2. Development Status of Relevant YANG Models	7
3.6. NAT Management	8
3.6.1. Requirements	8
3.6.2. Development Status of Relevant YANG Models	8
3.7. IPv6 Transition Mechanisms Management	8
3.7.1. Requirements	8
3.7.2. Development of Relevant YANG Models	9
3.8. Management of Specific Services	9
3.8.1. Requirements	9
3.8.2. Development of Relevant YANG Models	9
3.9. Management of Security Components	10
3.9.1. Requirements	10
3.9.2. Development of Relevant YANG Models	10
3.10. Remote CPE Software Upgrade	10
3.10.1. Requirements	10
3.10.2. Development of Relevant YANG Models	11
4. Security Considerations	11
5. IANA Considerations	11
6. Acknowledgements	11
7. References	11
7.1. Normative References	11
7.2. Informative References	14
Authors' Addresses	15

1. Introduction

This document defines the requirements and specifies the necessary YANG models for managing residential CPE devices using NETCONF and YANG. Implementing NETCONF on CPE devices, along with the relevant YANG models, provides operators with a flexible and extensible management interface.

Many of the YANG models referenced here are in various stages in the development process. In some cases there is currently no existing work. The aim of this document is to catalog which models are necessary, and for each referenced YANG model, provide information about the current status of the existing work. It is intended as a 'living document', which will be updated as the required / referenced YANG models progress. Once finalised, the goal of the document is to serve as a CPE YANG 'Device profile' that can be used as a reference for operators and implementors who are adding YANG management capabilities to their devices.

2. Terminology

CPE	Customer Premises Equipment; provides access between a customer's LAN connected devices and their ISP's network. In the context of this document, the CPE device implements NETCONF/YANG. This document focuses on the type of residential CPE that typically exists between the Internet Service Provider access line and residential customer home, doing similar functions that for example [RFC7084] lists.
Existing RFCs	Lists YANG models defined in published RFCs.
Work In Progress	YANG models under development in active Internet Drafts, or relevant documents being produced by SDOs other than the IETF.
To Be Defined	YANG models that are identified as necessary for CPE management, but are not currently known to be in development at the time of writing.

3. Management Requirements

3.1. Interfaces

A CPE has a number of network interfaces, usually including some of the following interface types: Ethernet LAN, Ethernet WAN, Ethernet 802.1q, Ethernet 802.1ag, and WLAN (802.11a/b/n/g/ac). [RFC7223] defines a YANG model for general interface management, which identifies these (and other) interface types. However, Ethernet

standardisation is carried out by the IEEE, so it is probable where YANG models for managing these interfaces would be developed.

NB - The list of interface types necessary for a complete, general HGW model needs to include xDSL (BBF) and DOCSIS (ITU) interfaces. These will be included in a future version of this document.

3.1.1. Requirements

The following requirements are necessary for basic CPE management functionality.

INT-1: The CPE YANG implementation MUST implement general interface management.

INT-2: The CPE YANG implementation MUST enable the configuration and management for the following interface types:

- o Ethernet LAN
- o Ethernet 802.1q
- o Ethernet 802.1ag (including Ethernet CFM)
- o Ethernet WAN
- o WLAN (802.11a/b/n/g/ac)

INT-3: The CPE YANG implementation MUST provide support for optical parameter configuration for the Ethernet WAN interface YANG model.

3.1.2. Development Status of Relevant YANG Models

Existing RFCs:

- o YANG Data Model for Interface Management [RFC7223].
- o IANA Interface Type YANG Module [RFC7224].

Work In Progress:

- o IEEE 802.1q YANG Model [IEEE-ETH-YANG]
- o Common Interface Extension YANG Data Models: [I-D.wilton-netmod-intf-ext-yang].
- o Interface VLAN YANG Data Models: [I-D.wilton-netmod-intf-vlan-yang].

To Be Defined:

- o Ethernet WAN
- o Ethernet 802.1ag
- o Ethernet LAN
- o WLAN (802.11a/b/n/g/ac)

3.2. IP Management

3.2.1. Requirements

The following requirements are necessary for the management and configuration of IPv4 and IPv6.

IP-1: The CPE YANG implementation MUST enable the configuration and management of IPv4 addresses and associated parameters on L3 interfaces.

IP-2: The CPE YANG implementation MUST enable the configuration and management of IPv6 addresses and associated parameters on L3 interfaces.

3.2.2. Development of Relevant YANG Models

Existing RFCs:

- o YANG Data Model for IP Management [RFC7277].

Work In Progress:

- o YANG Model for DiffServ: [I-D.asechoud-netmod-diffserv-model].

To Be Defined:

- o None

3.3. Routing and Multicast Management

3.3.1. Requirements

The following requirements are necessary for routing management.

ROUT-1: The CPE YANG implementation MUST provide support for the configuration and management of relevant IPv4/IPv6 dynamic routing protocols (for instance the ones relevant to IETF HOMENET WG).

ROUT-2: The CPE YANG implementation MUST include YANG models for the management of static IPv4/IPv6 routes.

ROUT-3: The CPE YANG implementation MUST provide support for the management of Protocol Independent Multicast (PIM).

ROUT-4: The CPE YANG implementation MUST provide support for the management of static multicast routes.

3.3.2. Development of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o YANG Data Model for Routing Management: [I-D.ietf-netmod-routing-cfg].
- o YANG model for static IPv4/IPv6 route: Appendix B in [I-D.ietf-netmod-routing-cfg].
- o YANG Data Model for ISIS protocol: [I-D.ietf-isis-yang-isis-cfg].
- o YANG model for PIM: [I-D.mcallister-pim-yang].
- o YANG model for IGMP and MLD: [I-D.liu-pim-igmp-mld-yang].

To Be Defined:

- o Static Multicast Route
- o What is the HOMENET relevant dynamic routing protocol.

3.4. CPE NETCONF Server Management

3.4.1. Requirements

The following requirements are necessary for management of the CPE's NETCONF Server.

- NETCONF-1: The CPE YANG implementation MUST provide support for management and configuration of its local NETCONF server using the NETCONF protocol.
- NETCONF-2: The CPE YANG implementation MUST provide support for the base notification function in order to allow a NETCONF client to retrieve notifications for common system events.
- NETCONF-3: The CPE YANG implementation MUST be able to retrieve NETCONF server configuration automatically during the bootstrap process (ZeroTouch).
- NETCONF-4: The CPE YANG implementation as a NETCONF server MUST provide support for the Call Home function so that a secure connection to a NETCONF client can be initiated.

3.4.2. Development Status of Relevant YANG Models

Existing RFCs:

- o YANG Module for NETCONF Monitoring: [RFC6022].
- o NETCONF Base Notifications: [RFC6470].

Work In Progress:

- o ZeroTouch: [I-D.ietf-netconf-zerotouch].
- o NETCONF Call Home: [I-D.ietf-netconf-call-home].
- o NETCONF Server Configuration Models:
[I-D.ietf-netconf-server-model].

To Be Defined:

- o None

3.5. DHCP/SLAAC/ND Management

3.5.1. Requirements

The following requirements are necessary for management of DHCP, SLAAC and ND.

- V6CONF-1: The CPE YANG implementation MUST provide support for management of its DHCPv4 server, which typically runs at the IPv4 LAN side.
- V6CONF-2: The CPE YANG implementation MUST provide support for the management of its DHCPv6 server, which can run at the IPv6 LAN side.
- V6CONF-3: The CPE YANG implementation MUST provide support for the management of its DHCPv6 client, which typically runs at the IPv6 WAN side.
- V6CONF-4: The CPE YANG implementation MUST provide support for the management of its DHCPv6 Prefix Delegation configuration (as a requesting router).
- V6CONF-5: The CPE YANG implementation MUST provide support for the management of SLAAC for stateless IPv6 configuration.

3.5.2. Development Status of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o YANG models for DHCPv4: [I-D.liu-dhc-dhcp-yang-model].
- o YANG Data Model for DHCPv6 Configuration:
[I-D.cui-dhc-dhcpv6-yang].

To Be Defined:

- o YANG model for SLAAC (Router Advertisement)

- o YANG model for Neighbour Discovery Protocol (NDP)
- o YANG model for DHCPv6 Prefix Delegation (requesting router)
- o YANG model for IPCP.
- o YANG model for IPv6CP.

3.6. NAT Management

3.6.1. Requirements

The following requirements are necessary for NAT Management.

NAT-1: The CPE YANG implementation MUST provide support for management of NAT44 configuration, as well as NAPT44 configuration.

3.6.2. Development Status of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o YANG Data Model for NAT44 and stateful NAT64 function [I-D.sivakumar-yang-nat].

To Be Defined:

- o None

3.7. IPv6 Transition Mechanisms Management

3.7.1. Requirements

The following requirements are necessary for management of IPv6 Transition Mechanisms.

TRAN-2: The CPE YANG implementation must include configuration and management for 6rd [RFC5969].

TRAN-2: The CPE YANG implementation must include configuration and management for DS-Lite [RFC6333].

TRAN-3: The CPE YANG implementation must include configuration and management for Lightweight 4over6 [RFC7596].

TRAN-4: The CPE YANG implementation must include configuration and management for MAP-E [RFC7597].

TRAN-5: The CPE YANG implementation must include configuration and management for MAP-T [RFC7599].

3.7.2. Development of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o YANG model for IPv4-in-IPv6 Softwire: [I-D.sun-softwire-yang].
- o YANG Data Model for the DS-Lite Address Family Transition Router (AFTR): [I-D.boucadair-softwire-dslite-yang].

To Be Defined:

- o YANG model for 6rd.
- o DHCP 4o6 client: May be combined in DHCPv6 YANG model as a feature.
- o DNS64
- o Stateless NAT64 (required for MAP-T and 464xlat).

3.8. Management of Specific Services

3.8.1. Requirements

The following requirements are necessary for management of specific services which the CPE may offer.

- SERVICE-1: The CPE YANG implementation MUST provide support for the management of a SIP client.
- SERVICE-2: The CPE YANG implementation MUST provide support for the management of a the CPEs Web server (used to provide a local management interface).
- SERVICE-3: The CPE YANG implementation MUST provide support for the management of an NTP client and server.
- SERVICE-4: The CPE YANG implementation MUST provide support for the management of the SSH server.

3.8.2. Development of Relevant YANG Models

Existing RFCs:

- o NTP Client: [RFC7317]

Work In Progress:

- o None

To Be Defined:

- o SIP Client
- o Web server, used by the customer for configuring their CPE device.
- o NTP server
- o SSH server

3.9. Management of Security Components

3.9.1. Requirements

The following requirements are necessary for management of security components.

SEC-1: The CPE YANG implementation MUST provide support for the management of IPv4 firewall and ACL functions.

SEC-1: The CPE YANG implementation MUST provide support for the management of IPv6 firewall and ACL functions.

3.9.2. Development of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o IPv4 Firewall configuration: [I-D.ietf-netmod-acl-model]
- o IPv6 Firewall configuration: [I-D.ietf-netmod-acl-model]
- o Access Control List (ACL): [I-D.ietf-netmod-acl-model]

To Be Defined:

- o IPv4/v6 Firewall (if needed in addition to the above)
- o Parental controls

3.10. Remote CPE Software Upgrade

3.10.1. Requirements

The following requirements are necessary to perform remote CPE Software file transfer and software upgrades.

SWUPG-1: The CPE implementation must provide a YANG model for the upgrade of firmware and software packages in order to fix bugs, enable new features, and resolve security issues.

SWUPG-2: The CPE YANG implementation MUST enable RPCs for file transfer in order to retrieve files from an operator-managed data centre, or upload logging.

3.10.2. Development of Relevant YANG Models

Existing RFCs:

- o None

Work In Progress:

- o File transfer: [I-D.sf-netmod-file-transfer-yang]

To Be Defined:

- o YANG model for firmware upgrade RPCs

4. Security Considerations

A NETCONF/YANG managed CPE should follow the Section 3.9 for enabling and managing IPv4/IPv6 firewalls. Security considerations from the related documents should be followed.

5. IANA Considerations

There are no IANA considerations for this document.

6. Acknowledgements

The authors would like to thank xxx for their contributions to this work.

7. References

7.1. Normative References

[I-D.asechoud-netmod-diffserv-model]

Choudhary, A., Shah, S., Jethanandani, M., Liu, B., and N. Strahle, "YANG Model for Diffserv", draft-asechoud-netmod-diffserv-model-03 (work in progress), June 2015.

[I-D.boucadair-softwire-dslite-yang]

Boucadair, M., Jacquenet, C., and S. Sivakumar, "YANG Data Model for the DS-Lite Address Family Transition Router (AFTR)", draft-boucadair-softwire-dslite-yang-02 (work in progress), September 2015.

[I-D.cui-dhc-dhcpv6-yang]

Cui, Y., Wang, H., Sun, L., Lemon, T., and I. Farrer, "YANG Data Model for DHCPv6 Configuration", draft-cui-dhc-dhcpv6-yang-04 (work in progress), September 2015.

- [I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, J., and L. Lhotka, "YANG Data Model for ISIS protocol", draft-ietf-isis-yang-isis-cfg-06 (work in progress), September 2015.
- [I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-11 (work in progress), September 2015.
- [I-D.ietf-netconf-server-model]
Watsen, K. and J. Schoenwaelder, "NETCONF Server and RESTCONF Server Configuration Models", draft-ietf-netconf-server-model-08 (work in progress), October 2015.
- [I-D.ietf-netconf-zerotouch]
Watsen, K., Clarke, J., and M. Abrahamsson, "Zero Touch Provisioning for NETCONF Call Home (ZeroTouch)", draft-ietf-netconf-zerotouch-03 (work in progress), July 2015.
- [I-D.ietf-netmod-acl-model]
Bogdanovic, D., Sreenivasa, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-03 (work in progress), June 2015.
- [I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-19 (work in progress), May 2015.
- [I-D.liu-dhc-dhcp-yang-model]
Liu, B. and K. Lou, "A YANG Data Model for DHCP Configuration", draft-liu-dhc-dhcp-yang-model-01 (work in progress), July 2015.
- [I-D.liu-pim-igmp-mld-yang]
Liu, Y. and F. Guo, "Yang Model for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD)", draft-liu-pim-igmp-mld-yang-01 (work in progress), March 2015.
- [I-D.mcallister-pim-yang]
Liu, X., McAllister, P., and A. Peter, "A YANG data model for Protocol-Independent Multicast (PIM)", draft-mcallister-pim-yang-00 (work in progress), July 2015.

- [I-D.perrault-behave-natv2-mib]
Perreault, S., Tsou, T., Sivakumar, S., and T. Taylor,
"Definitions of Managed Objects for Network Address
Translators (NAT)", draft-perrault-behave-natv2-mib-05
(work in progress), June 2015.
- [I-D.sf-netmod-file-transfer-yang]
Sun, Q. and I. Farrer, "A YANG Data Model for Transferring
Files", draft-sf-netmod-file-transfer-yang-00 (work in
progress), March 2015.
- [I-D.sivakumar-yang-nat]
Sivakumar, S., Boucadair, M., and S. <>, "YANG Data Model
for Network Address Translation (NAT)", draft-sivakumar-
yang-nat-03 (work in progress), September 2015.
- [I-D.sun-softwire-yang]
Sun, Q., Wang, H., Cui, Y., Farrer, I., Boucadair, M., and
R. Asati, "YANG Data Model for IPv4-in-IPv6 Softwire",
draft-sun-softwire-yang-04 (work in progress), October
2015.
- [I-D.wilton-netmod-intf-ext-yang]
Wilton, R., Ball, D., Singh, T., and S. Sivaraj, "Common
Interface Extension YANG Data Models", draft-wilton-
netmod-intf-ext-yang-00 (work in progress), July 2015.
- [I-D.wilton-netmod-intf-vlan-yang]
Wilton, R., Ball, D., Singh, T., and S. Sivaraj,
"Interface VLAN YANG Data Models", draft-wilton-netmod-
intf-vlan-yang-00 (work in progress), July 2015.
- [IEEE-ETH-YANG]
"IEEE 802.1q YANG Model",
<<http://www.ieee802.org/1/files/public/docs2015/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6022] Scott, M. and M. Bjorklund, "YANG Module for NETCONF
Monitoring", RFC 6022, DOI 10.17487/RFC6022, October 2010,
<<http://www.rfc-editor.org/info/rfc6022>>.
- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF)
Base Notifications", RFC 6470, DOI 10.17487/RFC6470,
February 2012, <<http://www.rfc-editor.org/info/rfc6470>>.

- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

7.2. Informative References

- [RFC5969] Townsley, W. and O. Troan, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) -- Protocol Specification", RFC 5969, DOI 10.17487/RFC5969, August 2010, <<http://www.rfc-editor.org/info/rfc5969>>.
- [RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <<http://www.rfc-editor.org/info/rfc6333>>.
- [RFC7084] Singh, H., Beebe, W., Donley, C., and B. Stark, "Basic Requirements for IPv6 Customer Edge Routers", RFC 7084, DOI 10.17487/RFC7084, November 2013, <<http://www.rfc-editor.org/info/rfc7084>>.
- [RFC7596] Cui, Y., Sun, Q., Boucadair, M., Tsou, T., Lee, Y., and I. Farrer, "Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture", RFC 7596, DOI 10.17487/RFC7596, July 2015, <<http://www.rfc-editor.org/info/rfc7596>>.
- [RFC7597] Troan, O., Ed., Dec, W., Li, X., Bao, C., Matsushima, S., Murakami, T., and T. Taylor, Ed., "Mapping of Address and Port with Encapsulation (MAP-E)", RFC 7597, DOI 10.17487/RFC7597, July 2015, <<http://www.rfc-editor.org/info/rfc7597>>.
- [RFC7599] Li, X., Bao, C., Dec, W., Ed., Troan, O., Matsushima, S., and T. Murakami, "Mapping of Address and Port using Translation (MAP-T)", RFC 7599, DOI 10.17487/RFC7599, July 2015, <<http://www.rfc-editor.org/info/rfc7599>>.

Authors' Addresses

Ian Farrer
Deutsche Telekom AG
CTO-ATI, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: ian.farrer@telekom.de

Qi Sun
Deutsche Telekom AG
CTO-ATI, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: sunqi.ietf@gmail.com

Sladjana Zoric
Deutsche Telekom AG
CTO-IPT, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: sladjana.zoric@telekom.de

Mikael Abrahamsson
T-Systems

Email: mikael.abrahamsson@t-systems.se

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2016

D. Bogdanovic

K. Sreenivasa
Brocade Communications System
L. Huang
Juniper Networks
D. Blair
Cisco Systems
October 17, 2015

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-05

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 19, 2016.

Copyright Notice

Copyright (c) 2015 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	4
3.1. ACL Modules	4
4. ACL YANG Models	6
4.1. IETF Access Control List module	6
4.2. IETF-PACKET-FIELDS module	10
4.3. An ACL Example	15
4.4. Port Range Usage Example	17
5. Linux nftables	17
6. Security Considerations	18
7. IANA Considerations	18
8. Acknowledgements	19
9. References	19
9.1. Normative References	19
9.2. Informative References	20
Appendix A. Extending ACL model examples	20
A.1. Example of extending existing model for route filtering	20
A.2. A company proprietary module example	22
A.3. Attaching Access Control List to interfaces	25
A.4. Example to augment model with mixed ACL type	27
Authors' Addresses	27

1. Introduction

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

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

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

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

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

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

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

Access Control List is also widely known as ACL (pronounce as [ak-uh l]) or Access List. In this document, Access Control List, ACL and Access List are interchangeable.

1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

AFI: Address Field Identifier

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

TCP: Transmission Control Protocol

2. Problem Statement

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

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

3. Design of the ACL Model

Although different vendors have different ACL data models, there is a common understanding of what access control list (ACL) is. A network system usually have a list of ACLs, and each ACL contains an ordered list of rules, also known as access list entries - ACEs. Each ACE has a group of match criteria and a group of action criteria. The match criteria consist of packet header matching and metadata matching. Packet header matching applies to fields visible in the packet such as address or class of service or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs. The model also includes a container to hold overall operational state for each ACL and operational state for each ACE. One ACL can be applied to multiple targets within the device, such as interfaces of a networked device, applications or features running in the device, etc. When applied to interfaces of a networked device, the ACL is applied in a direction which indicates if it should be applied to packet entering (input) or leaving the device (output). An example in the appendix shows how to express it in YANG model.

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

3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-access-control-list", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-access-control-list" uses groupings in "ietf-packet-fields". If there is a need to define new "matches" choice, such as IPFIX [RFC5101], the container "matches" can be augmented.

```

module: ietf-access-control-list
  +--rw access-lists
    +--rw acl* [acl-type acl-name]
      +--ro acl-oper-data
      +--rw access-list-entries
        +--rw ace* [rule-name]
          +--rw matches
            +--rw (ace-type)?
              +--:(ace-ip)
                +--rw (ace-ip-version)?
                  +--:(ace-ipv4)
                    +--rw destination-ipv4-network?      inet:ipv4-pr
efix      |      |      |      |      |      |      +--rw source-ipv4-network?      inet:ipv4-pr
efix      |      |      |      |      |      |      +--:(ace-ipv6)
          |      |      |      |      |      |      +--rw destination-ipv6-network?  inet:ipv6-pr
efix      |      |      |      |      |      |      +--rw source-ipv6-network?      inet:ipv6-pr
efix      |      |      |      |      |      |      +--rw flow-label?                inet:ipv6-fl
ow-label |      |      |      |      |      |      +--rw dscp?                      inet:dscp
          |      |      |      |      |      |      +--rw protocol?                  uint8
          |      |      |      |      |      |      +--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
          |      |      |      |      |      |      +--:(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-time
          |      |      |      |      |      |      | +--rw start?      yang:date-and-time
          |      |      |      |      |      |      | +--rw end?        yang:date-and-time
          |      |      |      |      |      |      | +--rw active?    boolean
          |      |      |      |      |      |      +--rw actions
          |      |      |      |      |      |      | +--rw (packet-handling)?
          |      |      |      |      |      |      | +--:(deny)
          |      |      |      |      |      |      | | +--rw deny?      empty
          |      |      |      |      |      |      | +--:(permit)
          |      |      |      |      |      |      | | +--rw permit?    empty
          |      |      |      |      |      |      +--ro ace-oper-data
          |      |      |      |      |      |      | +--ro match-counter?  yang:counter64
          |      |      |      |      |      |      +--rw rule-name          string
          |      |      |      |      |      |      +--rw acl-name            string
          |      |      |      |      |      |      +--rw acl-type            acl-type

```

Figure 1

4. ACL YANG Models

4.1. IETF Access Control List module

"ietf-access-control-list" is the standard top level module for Access lists. The "access-lists" container stores a list of "acl". Each "acl" 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", has containers defining "matches" and "actions". The "matches" define criteria used to identify patterns in "ietf-packet-fields". The "actions" define behavior to undertake once a "match" has been identified.

```
<CODE BEGINS>file "ietf-access-control-list@2015-10-11.yang"
module ietf-access-control-list {
  yang-version 1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-access-control-list";
  prefix acl;
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-packet-fields {
    prefix packet-fields;
  }
  organization "IETF NETMOD (NETCONF Data Modeling Language)
    Working Group";
  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
    WG List: netmod@ietf.org
    WG Chair: Juergen Schoenwaelder
    j.schoenwaelder@jacobs-university.de
    WG Chair: Tom Nadeau
    tnadeau@lucidvision.com
    Editor: Dean Bogdanovic
    ivandean@gmail.com
    Editor: Kiran Agrahara Sreenivasa
    kkoushik@brocade.com
    Editor: Lisa Huang
    lyihuang@juniper.net
    Editor: Dana Blair
    dblair@cisco.com";
  description
    "This YANG module defines a component that describing the
    configuration of Access Control Lists (ACLs).
    Copyright (c) 2015 IETF Trust and the persons identified as
    the document authors. All rights reserved."
```

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

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

```
revision 2015-03-17 {
  description
    "Base model for Network Access Control List (ACL).";
  reference
    "RFC XXXX: Network Access Control List (ACL)
    YANG Data Model";
}
identity acl-base {
  description
    "Base Access Control List type for all Access Control List type
    identifiers.";
}
identity ipv4-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields from the IPv4 header
    (e.g. IPv4 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv4-acl does not contain
    matches on fields in the ethernet header or the IPv6 header.";
}
identity ipv6-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields from the IPv6 header
    (e.g. IPv6 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv6-acl does not contain
    matches on fields in the ethernet header or the IPv4 header.";
}
identity eth-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields in the ethernet header.
    An acl of type eth-acl does not contain matches on fields in
    the IPv4 header, IPv6 header or layer 4 headers.";
}
typedef acl-type {
  type identityref {
    base acl-base;
  }
  description
```

```
    "This type is used to refer to an Access Control List
    (ACL) type";
}
typedef access-control-list-ref {
  type leafref {
    path "/access-lists/acl/acl-name";
  }
  description
    "This type is used by data models that need to reference an
    Access Control List";
}
container access-lists {
  description
    "This is a top level container for Access Control Lists.
    It can have one or more Access Control Lists.";
  list acl {
    key "acl-type acl-name";
    description
      "An Access Control List(ACL) is an ordered list of
      Access List Entries (ACE). Each Access Control Entry has a
      list of match criteria and a list of actions.
      Since there are several kinds of Access Control Lists
      implemented with different attributes for
      different vendors, this
      model accommodates customizing Access Control Lists for
      each kind and for each vendor.";
    container acl-oper-data {
      config false;
      description
        "Overall Access Control List operational data";
    }
    container access-list-entries {
      description
        "The access-list-entries container contains
        a list of access-list-entries(ACE).";
      list ace {
        key "rule-name";
        ordered-by user;
        description
          "List of access list entries(ACE)";
        container matches {
          description
            "Definitions for match criteria for this Access List
            Entry.";
          choice ace-type {
            description
              "Type of access list entry.";
            case ace-ip {
```

```

        description "IP Access List Entry.";
    choice ace-ip-version {
        description
            "IP version used in this Access List Entry.";
        case ace-ipv4 {
            uses packet-fields:acl-ipv4-header-fields;
        }
        case ace-ipv6 {
            uses packet-fields:acl-ipv6-header-fields;
        }
    }
    uses packet-fields:acl-ip-header-fields;
}
case ace-eth {
    description
        "Ethernet Access List entry.";
    uses packet-fields:acl-eth-header-fields;
}
}
uses packet-fields:metadata;
}
container actions {
    description
        "Definitions of action criteria for this Access List
Entry.";
    choice packet-handling {
        default "deny";
        description
            "Packet handling action.";
        case deny {
            leaf deny {
                type empty;
                description
                    "Deny action.";
            }
        }
        case permit {
            leaf permit {
                type empty;
                description
                    "Permit action.";
            }
        }
    }
}
}
container ace-oper-data {
    config false;
    description

```

```

        "Operational data for this Access List Entry.";
    leaf match-counter {
        type yang:counter64;
        description
            "Number of matches for this Access List Entry";
    }
}
leaf rule-name {
    type string;
    description
        "A unique name identifying this Access List
        Entry(ACE).";
}
}
leaf acl-name {
    type string;
    description
        "The name of access-list. A device MAY restrict the length
        and value of this name, possibly space and special
        characters are not allowed.";
}
leaf acl-type {
    type acl-type;
    description
        "Type of access control list. Indicates the primary intended
        type of match criteria (e.g. ethernet, IPv4, IPv6, mixed, etc)
        used in the list instance.";
}
}
}
}
}
<CODE ENDS>

```

4.2. IETF-PACKET-FIELDS module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, transport layer fields and metadata. 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. In case more match conditions are needed, those can be added by augmenting choices within container "matches" in ietf-access-control-list.yang model

```

<CODE BEGINS>file "ietf-packet-fields@2015-06-11.yang"
module ietf-packet-fields {
    yang-version 1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";

```



```
prefix packet-fields;
import ietf-inet-types {
  prefix inet;
}
import ietf-yang-types {
  prefix yang;
}
organization "IETF NETMOD (NETCONF Data Modeling Language) Working
              Group";
contact
  "WG Web: http://tools.ietf.org/wg/netmod/
  WG List: netmod@ietf.org
  WG Chair: Juergen Schoenwaelder
            j.schoenwaelder@jacobs-university.de
  WG Chair: Tom Nadeau
            tnadeau@lucidvision.com
  Editor: Dean Bogdanovic
            deanb@juniper.net
  Editor: Kiran Agrahara Sreenivasa
            kkoushik@brocade.com
  Editor: Lisa Huang
            lyihuang@juniper.net
  Editor: Dana Blair
            dblair@cisco.com";
description
  "This YANG module defines groupings that are used by
  ietf-access-control-list YANG module. Their usage is not
  limited to ietf-access-control-list and can be
  used anywhere as applicable.
  Copyright (c) 2015 IETF Trust and the persons identified as
  the document authors. All rights reserved.
  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD
  License set forth in Section 4.c of the IETF Trust's Legal
  Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).
  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";
revision 2015-06-11 {
  description
    "Initial version of packet fields used by
    ietf-access-control-list";
  reference
    "RFC XXXX: Network Access Control List (ACL)
    YANG Data Model";
}
grouping acl-transport-header-fields {
```

```
description
  "Transport header fields";
container source-port-range {
  presence "Enables setting source port range";
  description
    "Inclusive range representing source ports to be used.
    When only lower-port is present, it represents a single port.";
  leaf lower-port {
    type inet:port-number;
    mandatory true;
    description
      "Lower boundary for port.";
  }
  leaf upper-port {
    must ". >= ../lower-port" {
      error-message
        "The upper-port must be greater than or equal to lower-port";
    }
    type inet:port-number;
    description
      "Upper boundary for port . If existing, the upper port
      must be greater or equal to lower-port.";
  }
}
container destination-port-range {
  presence "Enables setting destination port range";
  description
    "Inclusive range representing destination ports to be used. When
only lower-port is present, it represents a single port.";
  leaf lower-port {
    type inet:port-number;
    mandatory true;
    description
      "Lower boundary for port.";
  }
  leaf upper-port {
    must ". >= ../lower-port" {
      error-message
        "The upper-port must be greater than or equal to lower-port";
    }
    type inet:port-number;
    description
      "Upper boundary for port. If existing, the upper port must
      be greater or equal to lower-port.";
  }
}
}
grouping acl-ip-header-fields {
```

```
description
  "IP header fields common to ipv4 and ipv6";
leaf dscp {
  type inet:dscp;
  description
    "Value of dscp.";
}
leaf protocol {
  type uint8;
  description
    "Internet Protocol number.";
}
uses acl-transport-header-fields;
}
grouping acl-ipv4-header-fields {
  description
    "Fields in IPv4 header.";
  leaf destination-ipv4-network {
    type inet:ipv4-prefix;
    description
      "Destination IPv4 address prefix.";
  }
  leaf source-ipv4-network {
    type inet:ipv4-prefix;
    description
      "Source IPv4 address prefix.";
  }
}
grouping acl-ipv6-header-fields {
  description
    "Fields in IPv6 header";
  leaf destination-ipv6-network {
    type inet:ipv6-prefix;
    description
      "Destination IPv6 address prefix.";
  }
  leaf source-ipv6-network {
    type inet:ipv6-prefix;
    description
      "Source IPv6 address prefix.";
  }
  leaf flow-label {
    type inet:ipv6-flow-label;
    description
      "IPv6 Flow label.";
  }
}
reference
  "RFC 4291: IP Version 6 Addressing Architecture"
```

```
    RFC 4007: IPv6 Scoped Address Architecture
    RFC 5952: A Recommendation for IPv6 Address Text Representation";
}
grouping acl-eth-header-fields {
  description
    "Fields in Ethernet header.";
  leaf destination-mac-address {
    type yang:mac-address;
    description
      "Destination IEEE 802 MAC address.";
  }
  leaf destination-mac-address-mask {
    type yang:mac-address;
    description
      "Destination IEEE 802 MAC address mask.";
  }
  leaf source-mac-address {
    type yang:mac-address;
    description
      "Source IEEE 802 MAC address.";
  }
  leaf source-mac-address-mask {
    type yang:mac-address;
    description
      "Source IEEE 802 MAC address mask.";
  }
  reference
    "IEEE 802: IEEE Standard for Local and Metropolitan Area
    Networks: Overview and Architecture.";
}
grouping timerange {
  description
    "Time range contains time
    segments to allow access-control-list to be
    active/inactive when the system time
    is between the range.";
  container absolute-time {
    description
      "Absolute time and date that
      the associated function starts
      going into effect.";
    leaf start {
      type yang:date-and-time;
      description
        "Absolute start time and date";
    }
    leaf end {
      type yang:date-and-time;
    }
  }
}
```

```

        description
            "Absolute end time and date";
    }
    leaf active {
        type boolean;
        default "true";
        description
            "This object indicates whether the
            the ACL will be active(true) or
            inactive(false) during this time range.";
    }
}
}
grouping metadata {
    description
        "Fields associated with a packet which are not in
        the header.";
    leaf input-interface {
        type string;
        description
            "Packet was received on this interface.";
    }
    uses timerange;
}
}
<CODE ENDS>

```

4.3. An ACL Example

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

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

```

access-list ip sample-ip-acl
deny tcp host 10.10.10.1 host 10.10.10.255

```

Here is the example acl configuration xml:

```

<?xml version='1.0' encoding='UTF-8'?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <access-lists xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <acl-oper-data />
      <access-list-entries>
        <ace>
          <matches>

```

```
<destination-ipv4-network>
  10.10.10.255/24
</destination-ipv4-network>
<source-ipv4-network>
  10.10.10.1/24
</source-ipv4-network>
<destination-ipv6-network />
<source-ipv6-network />
<flow-label />
<dscp />
<protocol />
<source-port-range>
  <lower-port />
  <upper-port />
</source-port-range>
<destination-port-range>
  <lower-port />
  <upper-port />
</destination-port-range>
<destination-mac-address />
<destination-mac-address-mask />
<source-mac-address />
<source-mac-address-mask />
<input-interface />
<absolute-time>
  <start />
  <end />
</absolute-time>
</matches>
<actions>
  <deny />
  <permit />
</actions>
<ace-oper-data>
  <match-counter />
</ace-oper-data>
<rule-name>rule1<rule-name/>
</ace>
</access-list-entries>
<acl-name>sample-ipv4-acl<acl-name/>
<acl-type>ipv4-acl<acl-type/>
</acl>
</access-lists>
</data>
```

4.4. Port Range Usage Example

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

With the follow XML snippet:

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

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

With the follow XML snippet:

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

This represents source ports greater than/equal to 16384.

With the follow XML snippet:

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

This represents port 21.

5. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used (iptables, ip6tables, arptables, ebtables), so each one had separate data model. Recently, this has changed and a single utility, nftables, has been developed. With a single application, it has a single data model for firewall filters and it follows very similarly to the ietf-access-control list module proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

In the example below, it shows nftable configuration that accepts and count packets. It contains a

```
table ip filter {  
  chain output {  
    type filter hook output priority 0;  
    counter packets 1 bytes 84 accept  
  }  
}
```

There are many similarities between Linux nftables and IETF ACL YANG data models. It should be fairly easy to do translation between ACL YANG model described in this draft and Linux nftables.

6. Security Considerations

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

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

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

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

7. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:ietf-access-control-list

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

Registrant Contact: The IESG.

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

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

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

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

8. Acknowledgements

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

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

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder for their review of and suggestions to the draft.

9. References

9.1. Normative References

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

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

9.2. Informative References

- [RFC5101] Claise, B., Ed., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, DOI 10.17487/RFC5101, January 2008, <<http://www.rfc-editor.org/info/rfc5101>>.

Appendix A. Extending ACL model examples

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

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

```
file "ietf-example-ext-route-filter@2015-02-14.yang"
```

```
module ietf-example-ext-route-filter {  
  yang-version 1;  
  namespace "urn:ietf:params:xml:ns:yang:ietf-example-ext-route-filter";
```

```
prefix ietf-example-ext-route-filter;
import ietf-inet-types {
  prefix "inet";
}
import ietf-access-control-list {
  prefix "ietf-acl";
}
organization
  "Route modelele group.";

contact
  "abc@abc.com";

description "
  This module describes route filter as a collection of
  match prefixes. When specifying a match prefix, you
  can specify an exact match with a particular route or
  a less precise match. You can configure either a
  common action that applies to the entire list or an
  action associated with each prefix.
  ";
revision 2015-05-03 {
  description
    "Creating Route-Filter extension model based on
    ietf-access-control-list model";
  reference " ";
}
augment "/ietf-acl:access-lists/ietf-acl:acl/
  ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches"{
  description "
    This module augments the matches container in the ietf-acl
    module with route filter specific actions
    ";
  choice route-prefix{
    description "Define route filter match criteria";
    case range {
      description
        " Route falls between the lower prefix/prefix-length
        and the upperprefix/prefix-length.";
      choice ipv4-range {
        description "Defines the IPv4 prefix range";
        leaf v4-lower-bound {
          type inet:ipv4-prefix;
          description
            "Defines the lower IPv4 prefix/prefix length";
        }
        leaf v4-upper-bound {
          type inet:ipv4-prefix;

```

```

        description
            "Defines the upper IPv4 prefix/prefix length";
    }
}
choice ipv6-range {
    description "Defines the IPv6 prefix/prefix range";
    leaf v6-lower-bound {
        type inet:ipv6-prefix;
        description
            "Defines the lower IPv6 prefix/prefix length";
    }
    leaf v6-upper-bound {
        type inet:ipv6-prefix;
        description
            "Defines the upper IPv6 prefix/prefix length";
    }
}
}
}
}
}
```

A.2. A company proprietary module example

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

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

```

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

file "newco-acl@2015-03-04.yang"

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

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

  prefix example-newco-acl;

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

  revision 2015-05-03{
    description "Creating NewCo proprietary extensions to ietf-acl model";
  }

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

```

```
    }
  }

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

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

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

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

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

A.3. Attaching Access Control List to interfaces

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

This is an example of attaching an Access Control List to an interface.

```
import ietf-access-control-list {
  prefix "ietf-acl";
}
import ietf-interface {
  prefix "ietf-if";
}
import ietf-yang-types {
  prefix "yang";
}

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

augment "/ietf-acl:access-lists/ietf-acl:acl/ietf-acl:acl-oper-data" {
  description
    "This is an example on how to apply acl to a target to collect
    operational data";
  container targets{
    choice interface{
      leaf-list interface-name{
        type ietf-if:interface-ref;
      }
    }
  }
}
```


A.4. Example to augment model with mixed ACL type

As vendors (or IETF) add more features to ACL, the model is easily augmented. One of such augmentations can be to add support for mixed type of ACLs, where `acl-type-base` can be augmented like in example below:

```
identity mixed-l3-acl {
  base "access-control-list:acl-type-base";
  description "ACL that contains a mix of entries that
    primarily match on fields in IPv4 headers and entries
    that primarily match on fields in IPv6 headers.
    Matching on layer 4 header fields may also exist in the
    list. An acl of type mixed-l3-acl does not contain
    matches on fields in the ethernet header.";
}

identity mixed-l2-l3-acl {
  base "access-control-list:acl-type-base";
  description "ACL that contains a mix of entries that
    primarily match on fields in ethernet headers, entries
    that primarily match on fields in IPv4 headers, and entries
    that primarily match on fields in IPv6 headers. Matching on
    layer 4 header fields may also exist in the list.";
}
```

Authors' Addresses

Dean Bogdanovic

Email: ivandean@gmail.com

Kiran Agrahara Sreenivasa
Brocade Communications System

Email: kkoushik@brocade.com

Lisa Huang
Juniper Networks

Email: lyihuang@juniper.net

Dana Blair
Cisco Systems

Email: dblair@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

M. Bjorklund, Ed.
Tail-f Systems
October 19, 2015

The YANG 1.1 Data Modeling Language
draft-ietf-netmod-rfc6020bis-08

Abstract

YANG is a data modeling language used to model configuration data, state data, remote procedure calls, and notifications for network management protocols like the Network Configuration Protocol (NETCONF).

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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	10
3. Terminology	10
4. YANG Overview	13
4.1. Functional Overview	13
4.2. Language Overview	15
4.2.1. Modules and Submodules	15
4.2.2. Data Modeling Basics	15
4.2.3. State Data	19
4.2.4. Built-In Types	20
4.2.5. Derived Types (typedef)	21
4.2.6. Reusable Node Groups (grouping)	22
4.2.7. Choices	23
4.2.8. Extending Data Models (augment)	24
4.2.9. Operation Definitions	25
4.2.10. Notification Definitions	27
5. Language Concepts	28
5.1. Modules and Submodules	28
5.1.1. Import and Include by Revision	29
5.1.2. Module Hierarchies	30
5.2. File Layout	31
5.3. XML Namespaces	32
5.3.1. YANG XML Namespace	32
5.4. Resolving Grouping, Type, and Identity Names	32
5.5. Nested Typedefs and Groupings	32
5.6. Conformance	33
5.6.1. Basic Behavior	33
5.6.2. Optional Features	34
5.6.3. Deviations	34
5.6.4. Implementing a Module	35
5.6.5. Announcing Conformance Information in NETCONF	37
5.7. Datastore Modification	38
6. YANG Syntax	38

6.1.	Lexical Tokenization	39
6.1.1.	Comments	39
6.1.2.	Tokens	39
6.1.3.	Quoting	39
6.2.	Identifiers	41
6.2.1.	Identifiers and Their Namespaces	41
6.3.	Statements	42
6.3.1.	Language Extensions	42
6.4.	XPath Evaluations	42
6.4.1.	XPath Context	43
6.5.	Schema Node Identifier	45
7.	YANG Statements	45
7.1.	The module Statement	46
7.1.1.	The module's Substatements	46
7.1.2.	The yang-version Statement	47
7.1.3.	The namespace Statement	48
7.1.4.	The prefix Statement	48
7.1.5.	The import Statement	48
7.1.6.	The include Statement	49
7.1.7.	The organization Statement	50
7.1.8.	The contact Statement	50
7.1.9.	The revision Statement	50
7.1.10.	Usage Example	51
7.2.	The submodule Statement	52
7.2.1.	The submodule's Substatements	53
7.2.2.	The belongs-to Statement	53
7.2.3.	Usage Example	54
7.3.	The typedef Statement	54
7.3.1.	The typedef's Substatements	55
7.3.2.	The typedef's type Statement	55
7.3.3.	The units Statement	55
7.3.4.	The typedef's default Statement	55
7.3.5.	Usage Example	56
7.4.	The type Statement	56
7.4.1.	The type's Substatements	56
7.5.	The container Statement	56
7.5.1.	Containers with Presence	57
7.5.2.	The container's Substatements	57
7.5.3.	The must Statement	58
7.5.4.	The must's Substatements	59
7.5.5.	The presence Statement	60
7.5.6.	The container's Child Node Statements	60
7.5.7.	XML Encoding Rules	60
7.5.8.	NETCONF <edit-config> Operations	61
7.5.9.	Usage Example	61
7.6.	The leaf Statement	62
7.6.1.	The leaf's default value	62
7.6.2.	The leaf's Substatements	63

7.6.3.	The leaf's type Statement	63
7.6.4.	The leaf's default Statement	64
7.6.5.	The leaf's mandatory Statement	64
7.6.6.	XML Encoding Rules	64
7.6.7.	NETCONF <edit-config> Operations	64
7.6.8.	Usage Example	65
7.7.	The leaf-list Statement	66
7.7.1.	Ordering	66
7.7.2.	The leaf-list's default values	67
7.7.3.	The leaf-list's Substatements	67
7.7.4.	The leaf-list's default Statement	68
7.7.5.	The min-elements Statement	68
7.7.6.	The max-elements Statement	69
7.7.7.	The ordered-by Statement	69
7.7.8.	XML Encoding Rules	69
7.7.9.	NETCONF <edit-config> Operations	70
7.7.10.	Usage Example	71
7.8.	The list Statement	72
7.8.1.	The list's Substatements	72
7.8.2.	The list's key Statement	73
7.8.3.	The list's unique Statement	74
7.8.4.	The list's Child Node Statements	75
7.8.5.	XML Encoding Rules	75
7.8.6.	NETCONF <edit-config> Operations	76
7.8.7.	Usage Example	77
7.9.	The choice Statement	80
7.9.1.	The choice's Substatements	80
7.9.2.	The choice's case Statement	81
7.9.3.	The choice's default Statement	82
7.9.4.	The choice's mandatory Statement	84
7.9.5.	XML Encoding Rules	84
7.9.6.	NETCONF <edit-config> Operations	84
7.9.7.	Usage Example	84
7.10.	The anydata Statement	85
7.10.1.	The anydata's Substatements	86
7.10.2.	XML Encoding Rules	86
7.10.3.	NETCONF <edit-config> Operations	86
7.10.4.	Usage Example	87
7.11.	The anyxml Statement	87
7.11.1.	The anyxml's Substatements	88
7.11.2.	XML Encoding Rules	88
7.11.3.	NETCONF <edit-config> Operations	89
7.11.4.	Usage Example	89
7.12.	The grouping Statement	89
7.12.1.	The grouping's Substatements	90
7.12.2.	Usage Example	91
7.13.	The uses Statement	91
7.13.1.	The uses's Substatements	91

7.13.2.	The refine Statement	92
7.13.3.	XML Encoding Rules	93
7.13.4.	Usage Example	93
7.14.	The rpc Statement	94
7.14.1.	The rpc's Substatements	94
7.14.2.	The input Statement	95
7.14.3.	The output Statement	96
7.14.4.	NETCONF XML Encoding Rules	97
7.14.5.	Usage Example	97
7.15.	The action Statement	98
7.15.1.	The action's Substatements	98
7.15.2.	NETCONF XML Encoding Rules	99
7.15.3.	Usage Example	99
7.16.	The notification Statement	101
7.16.1.	The notification's Substatements	102
7.16.2.	NETCONF XML Encoding Rules	102
7.16.3.	Usage Example	103
7.17.	The augment Statement	104
7.17.1.	The augment's Substatements	105
7.17.2.	XML Encoding Rules	106
7.17.3.	Usage Example	106
7.18.	The identity Statement	108
7.18.1.	The identity's Substatements	108
7.18.2.	The base Statement	108
7.18.3.	Usage Example	109
7.19.	The extension Statement	109
7.19.1.	The extension's Substatements	110
7.19.2.	The argument Statement	110
7.19.3.	Usage Example	111
7.20.	Conformance-Related Statements	111
7.20.1.	The feature Statement	112
7.20.2.	The if-feature Statement	113
7.20.3.	The deviation Statement	114
7.21.	Common Statements	117
7.21.1.	The config Statement	117
7.21.2.	The status Statement	117
7.21.3.	The description Statement	118
7.21.4.	The reference Statement	118
7.21.5.	The when Statement	119
8.	Constraints	120
8.1.	Constraints on Data	120
8.2.	NETCONF Constraint Enforcement Model	121
8.2.1.	Payload Parsing	121
8.2.2.	NETCONF <edit-config> Processing	122
8.2.3.	Validation	123
9.	Built-In Types	123
9.1.	Canonical Representation	123
9.2.	The Integer Built-In Types	124

9.2.1.	Lexical Representation	124
9.2.2.	Canonical Form	125
9.2.3.	Restrictions	125
9.2.4.	The range Statement	125
9.2.5.	Usage Example	126
9.3.	The decimal64 Built-In Type	126
9.3.1.	Lexical Representation	126
9.3.2.	Canonical Form	127
9.3.3.	Restrictions	127
9.3.4.	The fraction-digits Statement	127
9.3.5.	Usage Example	128
9.4.	The string Built-In Type	128
9.4.1.	Lexical Representation	128
9.4.2.	Canonical Form	128
9.4.3.	Restrictions	128
9.4.4.	The length Statement	128
9.4.5.	The pattern Statement	129
9.4.6.	The modifier Statement	130
9.4.7.	Usage Example	130
9.5.	The boolean Built-In Type	131
9.5.1.	Lexical Representation	131
9.5.2.	Canonical Form	131
9.5.3.	Restrictions	132
9.6.	The enumeration Built-In Type	132
9.6.1.	Lexical Representation	132
9.6.2.	Canonical Form	132
9.6.3.	Restrictions	132
9.6.4.	The enum Statement	132
9.6.5.	Usage Example	133
9.7.	The bits Built-In Type	134
9.7.1.	Restrictions	134
9.7.2.	Lexical Representation	135
9.7.3.	Canonical Form	135
9.7.4.	The bit Statement	135
9.7.5.	Usage Example	136
9.8.	The binary Built-In Type	136
9.8.1.	Restrictions	136
9.8.2.	Lexical Representation	136
9.8.3.	Canonical Form	137
9.9.	The leafref Built-In Type	137
9.9.1.	Restrictions	137
9.9.2.	The path Statement	137
9.9.3.	The require-instance Statement	138
9.9.4.	Lexical Representation	138
9.9.5.	Canonical Form	138
9.9.6.	Usage Example	138
9.10.	The identityref Built-In Type	142
9.10.1.	Restrictions	142

9.10.2.	The identityref's base Statement	142
9.10.3.	Lexical Representation	143
9.10.4.	Canonical Form	143
9.10.5.	Usage Example	143
9.11.	The empty Built-In Type	145
9.11.1.	Restrictions	145
9.11.2.	Lexical Representation	145
9.11.3.	Canonical Form	145
9.11.4.	Usage Example	145
9.12.	The union Built-In Type	145
9.12.1.	Restrictions	146
9.12.2.	Lexical Representation	146
9.12.3.	Canonical Form	146
9.12.4.	Usage Example	146
9.13.	The instance-identifier Built-In Type	147
9.13.1.	Restrictions	148
9.13.2.	Lexical Representation	148
9.13.3.	Canonical Form	148
9.13.4.	Usage Example	148
10.	XPath Functions	149
10.1.	Functions for Node Sets	149
10.1.1.	current()	149
10.2.	Functions for Strings	149
10.2.1.	re-match()	149
10.3.	Functions for the YANG Types "leafref" and "instance- identifier"	150
10.3.1.	deref()	150
10.4.	Functions for the YANG Type "identityref"	151
10.4.1.	derived-from()	151
10.4.2.	derived-from-or-self()	151
10.5.	Functions for the YANG Type "enumeration"	152
10.5.1.	enum-value()	153
10.6.	Functions for the YANG Type "bits"	154
10.6.1.	bit-is-set()	154
11.	Updating a Module	154
12.	Coexistence with YANG version 1	157
13.	YIN	157
13.1.	Formal YIN Definition	157
13.1.1.	Usage Example	160
14.	YANG ABNF Grammar	161
15.	NETCONF Error Responses for YANG Related Errors	185
15.1.	Error Message for Data That Violates a unique Statement	185
15.2.	Error Message for Data That Violates a max-elements Statement	186
15.3.	Error Message for Data That Violates a min-elements Statement	186
15.4.	Error Message for Data That Violates a must Statement	186
15.5.	Error Message for Data That Violates a require-instance	

Statement	187
15.6. Error Message for Data That Does Not Match a leafref Type	187
15.7. Error Message for Data That Violates a mandatory choice Statement	187
15.8. Error Message for the "insert" Operation	187
16. IANA Considerations	188
17. Security Considerations	188
18. Contributors	188
19. Acknowledgements	189
20. ChangeLog	189
20.1. Version -08	189
20.2. Version -07	189
20.3. Version -06	189
20.4. Version -05	189
20.5. Version -04	189
20.6. Version -03	190
20.7. Version -02	190
20.8. Version -01	190
20.9. Version -00	191
21. References	191
21.1. Normative References	191
21.2. Informative References	192
Author's Address	193

1. Introduction

YANG is a data modeling language originally designed to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications [RFC6241]. Since the publication of YANG version 1 [RFC6020], YANG has been used or proposed to be used for other protocols (e.g., RESTCONF [I-D.ietf-netconf-restconf] and CoMI [I-D.vanderstok-core-comi]). Further, other encodings than XML have been proposed (e.g., JSON [I-D.ietf-netmod-yang-json]).

This document describes the syntax and semantics of version 1.1 of the YANG language. It also describes how a data model defined in a YANG module is encoded in the Extensible Markup Language (XML), and how NETCONF operations are used to manipulate the data. Other protocols and encodings are possible, but out of scope for this specification.

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.
- o Made noncharacters illegal in the built-in type "string".
- o Defined the legal characters in YANG modules.
- o Changed the rules for the interpretation of escaped characters in double quoted strings. This is an backwards incompatible change from YANG version 1. A module that uses a character sequence that is now illegal must change the string to match the new rules. See Section 6.1.3 for details.
- o Extended the "if-feature" syntax to be a boolean expression over feature names.
- o Allow "if-feature" in "bit", "enum", and "identity".
- o Allow "if-feature" in "refine".
- o Made "when" and "if-feature" illegal on list keys, unless the parent is also conditional, and the condition matches the parent's condition.
- o Allow "choice" as a shorthand case statement (see Section 7.9).
- o Added a new substatement "modifier" to pattern (see Section 9.4.6).
- o Allow "must" in "input", "output", and "notification".
- o Added a set of new XPath functions in Section 10.
- o Clarified the XPath context's tree in Section 6.4.1.
- o Defined the string value of an identityref in XPath expressions (see Section 9.10).
- o Clarified what unprefixed names mean in leafrefs in typedefs (see Section 9.9.2).
- o Allow identities to be derived from multiple base identities (see Section 7.18 and Section 9.10).
- o Allow enumerations to be subtyped (see Section 9.6).
- o Allow leaf-lists to have default values (see Section 7.7.2).

- o Use [RFC7405] syntax for case-sensitive strings in the grammar.
- o Changed the module advertisement mechanism (see Section 5.6.5).
- o Changed the scoping rules for definitions in submodules. A submodule can now reference all definitions in all submodules that belong to the same module, without using the "include" statement.
- o Added a new statement "action" that is used to define operations tied to data nodes.
- o Allow notifications to be tied to data nodes.
- o Added a new data definition statement "anydata" (see Section 7.10).

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

The following terms are used within this document:

- o action: An operation defined for a node in the data tree.
- o anydata: A data node that can contain an unknown set of nodes that can be modelled by YANG.
- 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 client: An entity that can access YANG-defined data on a server, over some network management protocol.

- o conformance: A measure of how accurately a server 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, anydata, 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, anydata, and anyxml.
- o data tree: An instantiated tree of any data modeled with YANG, e.g., configuration data, state data, combined configuration and state data, RPC or action input, RPC or action output, or notification.
- o derived type: A type that is derived from a built-in type (such as uint32), or another derived type.
- 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 servers that support that feature.
- o grouping: A reusable set of schema nodes, which may be used locally in the module 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 mandatory node: A mandatory node is one of:
 - * A leaf, choice, anydata, or anyxml node with a "mandatory" statement with the value "true".
 - * A list or leaf-list node with a "min-elements" statement with a value greater than zero.
 - * A container node without a "presence" statement, which has at least one mandatory node as a child.
- o module: A YANG module defines a hierarchy of schema nodes. 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.
- o RPC operation: A specific Remote Procedure Call.
- o schema node: A node in the schema tree. One of action, container, leaf, leaf-list, list, choice, case, rpc, input, output, notification, anydata, 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 server: An entity that provides access to YANG-defined data to a client, over some network management protocol.
- o server deviation: A failure of the server to implement a module faithfully.
- o submodule: A partial module definition that contributes derived types, groupings, data nodes, RPCs, actions, 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.

The following terms are defined in [RFC6241]:

- o configuration data
- o configuration datastore: a configuration datastore is an instantiated data tree with configuration data
- o datastore: an instantiated data tree
- o running configuration datastore
- o state data

4. YANG Overview

This non-normative section is intended to give a high-level overview of YANG to first-time readers.

4.1. Functional Overview

YANG is a language originally designed 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. Although out of scope for this specification, YANG can also be used with other protocols than NETCONF.

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 and used in either the same module or in another module that imports 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 13), 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 on-the-wire.

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 data models for network

management protocols such as NETCONF, not arbitrary XML documents or arbitrary data models.

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 [RFC6643]. However, YANG is not concerned with reverse translation from YANG to SMIV2.

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.

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 server may implement a number of modules, allowing multiple views of the same data, or multiple views of disjoint subsections of the server's data. Alternatively, the server may implement only one module that defines all available data.

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

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

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

4.2.2. Data Modeling Basics

YANG defines four types of data nodes for data modeling. In each of the following subsections, the examples show the YANG syntax as well as a corresponding XML encoding.

4.2.2.1. Leaf Nodes

A leaf instance 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";  
}
```

XML Encoding Example:

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

The "leaf" statement is covered in Section 7.6.

4.2.2.2. Leaf-List Nodes

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

YANG Example:

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

XML Encoding 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 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 (leaves, lists, containers, leaf-lists, and actions).

YANG Example:

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

XML Encoding 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;
  }
}
```

XML Encoding 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. In NETCONF, state data is reported using the <get> operation, not the <get-config> operation. Parent containers, lists, and key leaves are reported also, giving the context for the state data.

In this example, two leaves 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";  
  }  
}  
  
leaf completed {  
  type percent;  
}
```

XML Encoding 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;  
  }  
}
```

XML Encoding 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 7.12.

4.2.7. Choices

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

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

The choice and case nodes appear only in the schema tree but not in the data tree. 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;
        }
      }
    }
  }
}
```

XML Encoding 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 encoding 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:

XML Encoding Example:

```
<user>  
  <name>alice</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 7.17.

4.2.9. Operation Definitions

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

YANG Example:

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

NETCONF XML Example:

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

YANG Example:

```
list interface {
  key "name";

  leaf name {
    type string;
  }

  action ping {
    input {
      leaf destination {
        type inet:ip-address;
      }
    }
    output {
      leaf packet-loss {
        type uint8;
      }
    }
  }
}
```

NETCONF XML Example:

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interface xmlns="http://acme.example.com/system">
    <name>eth1</name>
    <ping>
      <destination>192.0.2.1</destination>
    </ping>
  </interface>
</rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:acme="http://acme.example.com/system">
  <acme:packet-loss>60</acme:packet-loss>
</rpc-reply>
```

The "rpc" statement is covered in Section 7.14, and the "action" statement in Section 7.15.

4.2.10. Notification Definitions

YANG allows the definition of notifications. 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 7.16.

5. Language Concepts

5.1. Modules and Submodules

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

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

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

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

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

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

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

- o For a module or submodule to reference definitions in an external module, the external module MUST be imported.
- o A module MUST include all its submodules.
- o A module or submodule belonging to that module can reference definitions in the module and all submodules included by the module.

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

When a definition in an external module is referenced, a locally defined prefix MUST be used, followed by ":", and then the external identifier. References to definitions in the local module MAY use the prefix notation. Since built-in data types do not belong to any module and have no prefix, references to built-in data types (e.g., int32) cannot use the prefix notation. The syntax for a reference to a definition is formally defined by the rule "identifier-ref" in Section 14.

5.1.1. Import and Include by Revision

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

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

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

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

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

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

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

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

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

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

5.1.2. Module Hierarchies

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

5.1.2.1. NETCONF XML Encoding

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 parsers can find imported modules and included submodules via this convention.

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.

XML namespaces for modules published in RFC streams [RFC4844] MUST be assigned by IANA, see section 14 in [RFC6020].

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

The "namespace" statement is covered in Section 7.1.3.

5.3.1. YANG XML Namespace

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

5.4. Resolving Grouping, Type, and Identity Names

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

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

5.5. Nested Typedefs and Groupings

Typedefs and groupings may appear nested under many YANG statements, allowing these to be lexically scoped by the hierarchy under which they appear. This allows types and groupings to be defined near where they are used, rather than placing them at the top level of the hierarchy. The close proximity increases readability.

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

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

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

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

5.6. Conformance

Conformance is a measure of how accurately a server follows the model. Generally speaking, servers are responsible for implementing the model faithfully, allowing applications to treat servers 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 a YANG-based 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 server controls whether these conditional portions of the model are supported or valid for that particular server.

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 server has local storage. If there is no local storage, an application should not tell the server 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 server. The model can express constructs that are not universally present in all servers. 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 server.

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 server supports a feature, then the corresponding portions of the module are valid for that server. If the server doesn't support the feature, those parts of the module are not valid, and applications should behave accordingly.

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

Further details are available in Section 7.20.1.

5.6.3. Deviations

In an ideal world, all servers would be required to implement the model exactly as defined, and deviations from the model would not be allowed. But in the real world, servers are often not able or designed to implement the model as written. For YANG-based automation to deal with these server deviations, a mechanism must exist for servers to inform applications of the specifics of such deviations.

For example, a BGP module may allow any number of BGP peers, but a particular server 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.

Server 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 server implementation deviates from the contract as defined in the module.

Further details are available in Section 7.20.3.

5.6.4. Implementing a Module

A server implements a module if it implements the module's data nodes, rpcs, actions, notifications, and deviations.

A server **MUST NOT** implement more than one revision of a module.

If a server implements a module A that imports a module B, and A uses any node from B in an "augment" or "path" statement that the server supports, then the server **MUST** implement a revision of module B that has these nodes defined. This is regardless of if module B is imported by revision or not.

If a server implements a module A that imports a module C without specifying the revision date of module C, and the server does not implement C (e.g., if C only defines some typedefs), the server **MUST** list module C in the "/modules/module" list from "ietf-yang-library" [I-D.ietf-netconf-yang-library], and it **MUST** set the leaf "conformance" to "import" for this module.

The reason for these rules is that clients need to be able to know the exact data model structure and types of all leafs and leaf-lists implemented in a server.

For example, with these modules:

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

  import b {
    revision-date 2015-01-01;
  }
}
```

```
import c;

revision 2015-01-01;

feature foo;

augment "/b:x" {
  if-feature foo;
  leaf y {
    type b:myenum;
  }
}

container a {
  leaf x {
    type c:bar;
  }
}
}

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

  revision 2015-04-04;
  revision 2015-01-01;

  typedef myenum {
    type enumeration {
      enum zero; // added in 2015-01-01
      enum one;  // added in 2015-04-04
    }
  }

  container x {           // added in 2015-01-01
    container y;          // added in 2015-04-04
  }
}

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

  revision 2015-03-03;
  revision 2015-02-02;
```

```
typedef foo {  
    ...  
}  
}
```

A server that implements revision "2015-01-01" of module "a" and supports feature "foo" can implement revision "2015-01-01" or "2015-04-04" of module "b". Since "b" was imported by revision, the type of leaf "/b:x/a:y" is the same regardless of which revision of "b" the server implements.

A server that implements module "a", but does not support feature "foo" does not have to implement module "b".

A server that implements revision "2015-01-01" of module "a" must pick a revision of module "c", and list it in the "/modules/module" list from "ietf-yang-library".

The following XML encoding example shows valid data for the "/modules/module" list for a server that implements module "a":

```
<modules xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">  
  <module-set-id>eelecb017370cafd</module-set-id>  
  <module>  
    <name>a</module>  
    <revision>2015-01-01</revision>  
    <feature>foo</feature>  
    <conformance>implement</conformance>  
  </module>  
  <module>  
    <name>b</module>  
    <revision>2015-04-04</revision>  
    <conformance>implement</conformance>  
  </module>  
  <module>  
    <name>c</module>  
    <revision>2015-02-02</revision>  
    <conformance>import</conformance>  
  </module>  
</modules>
```

5.6.5. Announcing Conformance Information in NETCONF

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

A NETCONF server announces the modules it implements by implementing the YANG module "ietf-yang-library" defined in [I-D.ietf-netconf-yang-library], and listing all implemented modules in the "/modules/module" list.

The server also advertises the following capability in the <hello> message (line-breaks and whitespaces are used for formatting reasons only):

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

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

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

5.7. Datastore Modification

Data models may allow the server to alter the configuration datastore in ways not explicitly directed via NETCONF protocol messages. For example, a data model may define leafs that are assigned system-generated values when the client does not provide one. A formal mechanism for specifying the circumstances where these changes are allowed is out of scope for this specification.

6. YANG Syntax

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

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

Legal characters in YANG modules are the Unicode and ISO/IEC 10646 [ISO.10646] characters, including tab, carriage return, and line feed but excluding the other C0 control characters, the surrogate blocks, and the noncharacters. The character syntax is formally defined by the rule "yang-char" in Section 14.

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

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

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

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, and **MAY** support longer identifiers. Identifiers are case sensitive. The identifier syntax is formally defined by the rule "identifier" in Section 14. 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, actions, notifications, anydatas, 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 7.19). 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.

The processing of extensions depends on whether support for those extensions is claimed for a given YANG parser or the tool set in which it is embedded. An unsupported extension, appearing in a YANG module as an unknown-statement (see Section 14) MAY be ignored in its entirety. Any supported extension MUST be processed in accordance with the specification governing that extension.

Care must be taken when defining extensions so that modules that use the extensions are meaningful also for applications that do not support the extensions.

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. An implementation is 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.

The data tree has no concept of document order. An implementation needs to choose some document order but how it is done is an implementation decision. This means that XPath expressions in YANG modules SHOULD not rely on any specific document order.

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

For example, consider the following definition:

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

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

6.4.1. XPath Context

All YANG XPath expressions share the following XPath context definition:

- o The set of namespace declarations is the set of all "import" statements' prefix and namespace pairs in the module where the XPath expression is specified, and the "prefix" statement's prefix for the "namespace" statement's URI.
- o Names without a namespace prefix belong to the same namespace as the identifier of the current node. Inside a grouping, that

namespace is affected by where the grouping is used (see Section 7.13). Inside a typedef, that namespace is affected by where the typedef is referenced. If a typedef is defined and referenced within a grouping, the namespace is affected by where the grouping is used (see Section 7.13).

- o The function library is the core function library defined in [XPath], and the functions defined in Section 10.
- o The set of variable bindings is empty.

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

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

- o If the XPath expression is defined in substatement to a data node that represents configuration, the accessible tree is the data in the datastore where the context node exists. The root node has all top-level configuration data nodes in all modules as children.
- o If the XPath expression is defined in a substatement to a data node that represents state data, the accessible tree is all state data in the server, and the running configuration datastore. The root node has all top-level data nodes in all modules as children.
- o If the XPath expression is defined in a substatement to a "notification" statement, the accessible tree is the notification instance, all state data in the server, and the running configuration datastore. The root node has the node representing the notification being defined and all top-level data nodes in all modules as children.
- o If the XPath expression is defined in a substatement to an "input" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance, all state data in the server, and the running configuration datastore. The root node has the node representing the operation being defined and all top-level data nodes in all modules as children. The node representing the operation being defined has the operation's input parameters as children.
- o If the XPath expression is defined in a substatement to an "output" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation's output, all state

data in the server, and the running configuration datastore. The root node has the node representing the operation being defined and all top-level data nodes in all modules as children. The node representing the operation being defined has the operation's output parameters as children.

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

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

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

6.5. Schema Node Identifier

A schema node identifier is a string that identifies a node in the schema tree. It has two forms, "absolute" and "descendant", defined by the rules "absolute-schema-nodeid" and "descendant-schema-nodeid" in Section 14, 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 in [RFC6020].

Private module names are assigned by the organization owning the module without a central registry. See Section 5.1 for recommendations on how to name modules.

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
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	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	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	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.

For compatibility between YANG version 1 and 1.1, see Section 12.

7.1.3. The namespace Statement

The "namespace" statement defines the XML namespace that all identifiers defined by the module are qualified by in the XML encoding, with the exception of identifiers for data nodes, action nodes, and notification nodes defined inside a grouping (see Section 7.13 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 can be imported, provided that different prefixes are used.

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.

7.1.6. The include Statement

The "include" statement is used to make content from a submodule available to that submodule's parent module. The argument is an identifier that is the name of the submodule to include. Modules are

only allowed to include submodules that belong to that module, as defined by the "belongs-to" statement (see Section 7.2.2).

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

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

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

Multiple revisions of the same submodule MUST NOT be included.

substatement	section	cardinality
revision-date	7.1.5.1	0..1

The includes's Substatements

7.1.7. The organization Statement

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

7.1.8. The contact Statement

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

7.1.9. The revision Statement

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

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

7.1.9.1. The revision's Substatement

substatement	section	cardinality
description	7.21.3	0..1
reference	7.21.4	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 in [RFC6020].

Private submodule names are assigned by the organization owning the submodule without a central registry. See Section 5.1 for recommendations on how to name submodules.

A submodule typically has the following layout:

```
submodule <module-name> {  
    <yang-version statement>  
  
    // module identification  
    <belongs-to statement>  
  
    // linkage statements  
    <import statements>  
  
    // meta information  
    <organization statement>  
    <contact statement>  
    <description statement>  
    <reference statement>  
  
    // revision history  
    <revision statements>  
  
    // module definitions  
    <other statements>  
}
```

7.2.1. The submodule's Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	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	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	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	7.16	0..n
organization	7.1.7	0..1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

7.2.2. The belongs-to Statement

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

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

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

substatement	section	cardinality
prefix	7.1.4	1

The belongs-to's Substatements

7.2.3. Usage Example

```

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

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

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

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

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

  description
    "This submodule defines common ACME types.";

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

  // definitions follows...
}

```

7.3. The typedef Statement

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

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

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

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

7.3.1. The typedef's Substatements

substatement	section	cardinality
default	7.3.4	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	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	7.18.2	0..n
bit	9.7.4	0..n
enum	9.6.4	0..n
fraction-digits	9.3.4	0..1
length	9.4.4	0..1
path	9.9.2	0..1
pattern	9.4.5	0..n
range	9.2.4	0..1
require-instance	9.9.3	0..1
type	7.4	0..n

7.5. The container Statement

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

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

7.5.1. Containers with Presence

YANG supports two styles of containers, those that exist only for organizing the hierarchy of data nodes, and those whose presence in the data tree 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 carries some meaning, representing a single bit of data.

In 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 server using ssh, but can also contain any ssh-related configuration knobs, such as connection rates or retry limits.

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

7.5.2. The container's Substatements

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
notification	7.16	0..n
presence	7.5.5	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	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 node in the accessible tree (see Section 6.4.1).

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

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

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

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

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

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

7.5.4. The must's Substatements

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	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> in NETCONF.

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> in NETCONF.

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", "anydata", and "anyxml" statements can be used to define child nodes to the container.

7.5.7. XML Encoding 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 or action 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.

If a non-presence container does not have any child nodes, the container may or may not be present in the XML encoding.

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 (see Section 7.5.1):

- 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	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	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 Encoding 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.

See Section 7.6.8 for an example.

7.6.7. NETCONF <edit-config> Operations

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

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

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

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

7.6.8. Usage Example

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

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

A corresponding XML instance example:

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

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

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

7.7. The leaf-list Statement

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

The values in a leaf-list MUST be unique.

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

7.7.1. Ordering

YANG supports two styles for ordering the entries within lists and leaf-lists. In many lists, the order of list entries does not impact the implementation of the list's configuration, and the server is free to sort the list entries in any reasonable order. The "description" string for the list may suggest an order to the server 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 server 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 packet filters entries are applied to incoming traffic may affect how that traffic is filtered. The user would need to decide if the filter entry that discards all TCP traffic should be applied before or after the filter entry that allows all traffic from trusted interfaces. The choice of order would be crucial.

YANG provides a rich set of facilities within NETCONF's <edit-config> operation that allows the order of list entries in user-ordered lists to be controlled. List entries may be inserted or rearranged, positioned as the first or last entry in the list, or positioned before or after another specific entry.

The "ordered-by" statement is covered in Section 7.7.7.

7.7.2. The leaf-list's default values

The default values of a leaf-list are the values that the server uses if the leaf-list does not exist in the data tree. The usage of the default values depends on the leaf-list's closest ancestor node in the schema tree that is not a non-presence-container (see Section 7.5.1):

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

In these cases, the default values are said to be in use.

When the default values are in use, the server MUST operationally behave as if the leaf-list was present in the data tree with the default values as its values.

If a leaf-list has one or more "default" statement, the leaf-list's default value are the values of the "default" statements, and if the leaf-list is user-ordered, the default values are used in the order of the "default" statements. Otherwise, if the leaf-list's type has a default value, and the leaf-list does not have a "min-elements" statement with a value greater than or equal to one, then the leaf-list's default value is the type's default value. In all other cases, the leaf-list does not have any default values.

7.7.3. The leaf-list's Substatements

substatement	section	cardinality
config	7.21.1	0..1
default	7.7.4	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.4	1
units	7.3.3	0..1
when	7.21.5	0..1

7.7.4. The leaf-list's default Statement

The "default" statement, which is optional, takes as an argument a string that contains a default value for the leaf-list.

The value of the "default" statement MUST be valid according to the type specified in the leaf-list's "type" statement.

The "default" statement MUST NOT be present on nodes where "min-elements" has a value greater than or equal to one.

7.7.5. The min-elements Statement

The "min-elements" statement, which is optional, takes as an argument a non-negative integer that puts a constraint on valid list entries. A valid leaf-list or list MUST have at least min-elements entries.

If no "min-elements" statement is present, it defaults to zero.

The behavior of the constraint depends on the type of the leaf-list's or list's closest ancestor node in the schema tree that is not a non-presence-container (see Section 7.5.1):

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

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

7.7.6. The max-elements Statement

The "max-elements" statement, which is optional, takes as an argument a positive integer or the string "unbounded", which puts a constraint on valid list entries. A valid leaf-list or list always has at most max-elements entries.

If no "max-elements" statement is present, it defaults to "unbounded".

The "max-elements" constraint is enforced according to the rules in Section 8.

7.7.7. The ordered-by Statement

The "ordered-by" statement defines whether the order of entries within a list are determined by the user or the system. The argument is one of the strings "system" or "user". If not present, order defaults to "system".

This statement is ignored if the list represents state data, RPC output parameters, or notification content.

See Section 7.7.1 for additional information.

7.7.7.1. ordered-by system

The entries in the list are sorted according to an order determined by the system. The "description" string for the list may suggest an order to the server implementor. If not, an implementation is free to sort the entries in the most appropriate order. An implementation SHOULD use the same order for the same data, regardless of how the data were created. Using a deterministic order will make comparisons possible using simple tools like "diff".

This is the default order.

7.7.7.2. ordered-by user

The entries in the list are sorted according to an order defined by the user. This order is controlled by using special XML attributes in the <edit-config> request. See Section 7.7.9 for details.

7.7.8. XML Encoding 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 or action input or output parameters.

See Section 7.7.10 for an example.

7.7.9. NETCONF <edit-config> Operations

Leaf-list entries can be created and deleted, but not modified, through <edit-config>, by using the "operation" attribute in the leaf-list entry's XML element.

In an "ordered-by user" leaf-list, the attributes "insert" and "value" in the YANG XML namespace (Section 5.3.1) can be used to control where in the leaf-list the entry is inserted. These can be used during "create" operations to insert a new leaf-list entry, or during "merge" or "replace" operations to insert a new leaf-list entry or move an existing one.

The "insert" attribute can take the values "first", "last", "before", and "after". If the value is "before" or "after", the "value" attribute MUST also be used to specify an existing entry in the leaf-list.

If no "insert" attribute is present in the "create" operation, it defaults to "last".

If several entries in an "ordered-by user" leaf-list are modified in the same <edit-config> request, the entries are modified one at the time, in the order of the XML elements in the request.

In a <copy-config>, or an <edit-config> with a "replace" operation that covers the entire leaf-list, the leaf-list order is the same as the order of the XML elements in the request.

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

If the operation is "merge" or "replace", the leaf-list entry is created if it does not exist.

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

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

7.7.10. Usage Example

```
leaf-list allow-user {  
    type string;  
    description  
        "A list of user name patterns to allow";  
}
```

A corresponding XML instance example:

```
<allow-user>alice</allow-user>  
<allow-user>bob</allow-user>
```

To create a new element in this list, using the default <edit-config> operation "merge":

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

Given the following ordered-by user leaf-list:

```
leaf-list cipher {  
    type string;  
    ordered-by user;  
    description  
        "A list of ciphers";  
}
```

The following would be used to insert a new cipher "blowfish-cbc" after "3des-cbc":

```
<rpc message-id="101"  
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"  
    xmlns:yang="urn:ietf:params:xml:ns:yang:1">  
  <edit-config>  
    <target>  
      <running/>  
    </target>  
    <config>  
      <system xmlns="http://example.com/schema/config">  
        <services>  
          <ssh>  
            <cipher nc:operation="create"  
                yang:insert="after"  
                yang:value="3des-cbc">blowfish-cbc</cipher>  
          </ssh>  
        </services>  
      </system>  
    </config>  
  </edit-config>  
</rpc>
```

7.8. The list Statement

The "list" statement is used to define an interior data node in the schema tree. A list node may exist in multiple instances in the data tree. Each such instance is known as a list entry. The "list" statement takes one argument, which is an identifier, followed by a block of substatements that holds detailed list information.

A list entry is uniquely identified by the values of the list's keys, if defined.

7.8.1. The list's Substatements

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
key	7.8.2	0..1
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
notification	7.16	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
unique	7.8.3	0..n
uses	7.13	0..n
when	7.21.5	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.

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 14.

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 14). 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 14.

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", "anydata", and "anyxml" statements can be used to define child nodes to the list.

7.8.5. XML Encoding 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 or action 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 or action 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 "first-name surname";

  leaf first-name {
    type string;
  }
  leaf surname {
    type string;
  }
  leaf type {
    type string;
  }
}
```

The following would be used to insert a new user "barney rubble" after the user "fred flintstone":

```
<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:first-name='fred']
            [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
          <type>admin</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

The following would be used to move the user "barney rubble" before the user "fred flintstone":

```
<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']
            [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
        </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 8.2.2 for additional information.

7.9.1. The choice's Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
default	7.9.3	0..1
description	7.21.3	0..1
if-feature	7.20.2	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	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	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 "anydata", "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 "anydata", "anyxml", "choice", "container", "leaf", "list", or "leaf-list" statement. In this case, the case node still

exists in the schema tree, and its identifier is the same as the identifier in the branch statement. Schema node identifiers (Section 6.5) MUST always explicitly include case node identifiers. 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
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
uses	7.13	0..n
when	7.21.5	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 statements of nodes under the cases (i.e., default values of leafs and leaf-lists, and default cases of nested choices). The default values and nested default cases 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) 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 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.

7.9.5. XML Encoding 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 or action 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 anydata Statement

The "anydata" 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 anydata information.

The "anydata" statement is used to represent an unknown set of nodes that can be modelled with YANG. An example of where this can be

useful is a list of received notifications, where the exact notifications are not known at design time.

An anydata node cannot be augmented (see Section 7.17).

An anydata node exists in zero or one instances in the data tree.

An implementation may or may not know the data model used to model a specific instance of an anydata node.

Since the use of anydata limits the manipulation of the content, it is RECOMMENDED that the "anydata" statement not be used to define configuration data.

7.10.1. The anydata's Substatements

substatement	section	cardinality
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.10.2. XML Encoding Rules

An anydata node is encoded as an XML element. The element's local name is the anydata's identifier, and its namespace is the module's XML namespace (see Section 7.1.3). The value of the anydata node is a set of nodes, which are encoded as XML subelements to the anydata element.

7.10.3. NETCONF <edit-config> Operations

An anydata 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 anydata node are ignored by the NETCONF server.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the anydata 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 subelements of the anydata 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 subelements of the anydata 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 "anydata" statement:

```
list logged-notification {  
  key time;  
  leaf time {  
    type yang:date-and-time;  
  }  
  anydata data;  
}
```

The following is a valid encoding of such a list instance:

```
<logged-notification>  
  <time>2014-07-29T13:43:12Z</time>  
  <data>  
    <notification  
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">  
      <eventTime>2014-07-29T13:43:01Z</eventTime>  
      <event xmlns="http://example.com/event">  
        <event-class>fault</event-class>  
        <reporting-entity>  
          <card>Ethernet0</card>  
        </reporting-entity>  
        <severity>major</severity>  
      </event>  
    </notification>  
  </data>
```

7.11. 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 in NETCONF.

An anyxml node cannot be augmented (see Section 7.17).

An anyxml node exists in zero or one instances in the data tree.

Since the use of anyxml limits the manipulation of the content, it is RECOMMENDED that the "anyxml" statement not be used to define configuration data.

It should be noted that in YANG version 1, anyxml was the only statement that could model an unknown hierarchy of data. In many cases this unknown hierarchy of data is actually modelled in YANG, but the exact YANG model is not known at design time. In these situations, it is RECOMMENDED to use anydata (Section 7.10) instead of anyxml.

7.11.1. The anyxml's Substatements

substatement	section	cardinality
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.11.2. XML Encoding 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 XML prefixes used in the encoding are local to each instance encoding. This means that the same XML may be encoded differently by different implementations.

7.11.3. 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.

7.11.4. Usage Example

Given the following "anyxml" statement:

```
anyxml html-info;
```

The following are two valid encodings of the same anyxml value:

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

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

7.12. The grouping Statement

The "grouping" statement is used to define a reusable block of nodes, which may be used locally in the module or submodule, and by other modules that import from it, according to the rules in Section 5.5.

It takes one argument, which is an identifier, followed by a block of substatements that holds detailed grouping information.

The "grouping" statement is not a data definition statement and, as such, does not define any nodes in the schema tree.

A grouping is like a "structure" or a "record" in conventional programming languages.

Once a grouping is defined, it can be referenced in a "uses" statement (see Section 7.13). 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 defined as direct children to a "grouping" statement are applied to the grouping itself.

7.12.1. The grouping's Substatements

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.12.2. 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;
  }
}

```

7.13. 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.

7.13.1. The uses's Substatements

substatement	section	cardinality
augment	7.17	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
refine	7.13.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.13.2. 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 A leaf-list node may get a set of default values, or a new set of default values if it already had defaults; i.e., the set of refined default values replaces the defaults already given.
- 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, anydata, 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, anydata, or anyxml node may get additional "must" expressions.
- o A leaf-list or list node may get a different "min-elements" or "max-elements" statement.
- o A leaf, leaf-list, list, container, anydata, or anyxml node may get additional "if-feature" expressions.
- o Any node can get refined extensions, if the extension allows refinement. See Section 7.19 for details.

7.13.3. XML Encoding 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.

7.13.4. Usage Example

To use the "endpoint" grouping defined in Section 7.12.2 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;
  }
}

```

7.14. The rpc Statement

The "rpc" statement is used to define an RPC operation. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed rpc information. This argument is the name of the RPC.

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.

7.14.1. The rpc's Substatements

substatement	section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n

7.14.2. The input Statement

The "input" statement, which is optional, is used to define input parameters to the operation. It does not take an argument. The substatements to "input" define nodes under the operation's input node.

If a leaf in the input tree has a "mandatory" statement with the value "true", the leaf **MUST** be present in an RPC invocation.

If a leaf in the input tree has a default value, the 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 RPC invocation with the default value as its value.

If a leaf-list in the input tree has one or more default values, the server **MUST** use these values in the same cases as described in Section 7.7.2. In these cases, the server **MUST** operationally behave as if the leaf-list was present in the RPC invocation with the default values as its values.

If a "config" statement is present for any node in the input tree, the "config" statement is ignored.

If any node has a "when" statement that would evaluate to false, then this node **MUST NOT** be present in the input tree.

7.14.2.1. The input's Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.3. The output Statement

The "output" statement, which is optional, is used to define output parameters to the RPC operation. It does not take an argument. The substatements to "output" define nodes under the operation's output node.

If a leaf in the output tree has a "mandatory" statement with the value "true", the leaf **MUST** be present in an RPC reply.

If a leaf in the output tree has a default value, the 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 RPC reply with the default value as its value.

If a leaf-list in the output tree has one or more default values, the client **MUST** use these values in the same cases as described in Section 7.7.2. In these cases, the client **MUST** operationally behave as if the leaf-list was present in the RPC reply with the default values as its values.

If a "config" statement is present for any node in the output tree, the "config" statement is ignored.

If any node has a "when" statement that would evaluate to false, then this node **MUST NOT** be present in the output tree.

7.14.3.1. The output's Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.4. NETCONF XML Encoding Rules

An rpc node is encoded as a child XML element to the <rpc> element, as designated by the substitution group "rpcOperation" 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.

7.14.5. 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>
```

7.15. The action Statement

The "action" statement is used to define an operation connected to a specific container or list data node. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed action information. The argument is the name of the action.

The "action" statement defines an action node in the schema tree. Under the action node, a schema node with the name "input", and a schema node with the name "output" are also defined. The nodes "input" and "output" are defined in the module's namespace.

An action MUST NOT be defined within an rpc, another action or a notification, i.e., an action node MUST NOT have an rpc, action, or a notification node as one of its ancestors in the schema tree. For example, this means that it is an error if a grouping that contains an action somewhere in its node hierarchy is used in a notification definition.

Since an action cannot be defined at the top-level of a module or in a case statement, it is an error if a grouping that contains an action at the top of its node hierarchy is used at the top-level of a module or in a case definition.

The difference between an action and an rpc is that an action is tied to a node in the datastore, whereas an rpc is not. When an action is invoked, the node in the datastore is specified along with the name of the action and the input parameters.

7.15.1. The action's Substatements

substatement	section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n

7.15.2. NETCONF XML Encoding Rules

When an action is invoked, an element with the local name "action" in the namespace "urn:ietf:params:xml:ns:yang:1" (see Section 5.3.1) is encoded as a child XML element to the <rpc> element defined in [RFC6241], as designated by the substitution group "rpcOperation" in [RFC6241].

The "action" element contains an hierarchy of nodes that identifies the node in the datastore. It MUST contain all containers and list nodes from the top level down to the list or container containing the action. For lists, all key leafs MUST also be included. The last container or list contains an XML element that carries the name of the defined action. Within this element the input parameters are encoded as child XML elements, in the same order as they are defined within the "input" statement.

Only one action can be invoked in one <rpc>. If more than one action is present in the <rpc>, the server MUST reply with an "bad-element" error-tag in the <rpc-error>.

If the action operation invocation succeeded, and no output parameters are returned, the <rpc-reply> contains a single <ok/> element defined in [RFC6241]. If output parameters are returned, they are encoded as child elements to the <rpc-reply> element defined in [RFC6241], in the same order as they are defined within the "output" statement.

7.15.3. Usage Example

The following example defines an action to reset one server at a server farm:

```
module server-farm {
  yang-version 1.1;
  namespace "http://example.net/server-farm";
  prefix "sfarm";

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

  list server {
    key name;
    leaf name {
      type string;
    }
    action reset {
      input {
        leaf reset-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

A corresponding XML instance example of the complete rpc and rpc-reply:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <server xmlns="http://example.net/server-farm">
      <name>apache-1</name>
      <reset>
        <reset-at>2014-07-29T13:42:00Z</reset-at>
      </reset>
    </server>
  </action>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <reset-finished-at xmlns="http://example.net/server-farm">
    2014-07-29T13:42:12Z
  </reset-finished-at>
</rpc-reply>
```

7.16. The notification Statement

The "notification" statement is used to define a 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.

A notification can be defined at the top-level of a module, or connected to a specific container or list data node in the schema tree.

A notification **MUST NOT** be defined within an `rpc`, `action` or another notification, i.e., a notification node **MUST NOT** have an `rpc`, `action`, or a notification node as one of its ancestors in the schema tree. For example, this means that it is an error if a grouping that contains an notification somewhere in its node hierarchy is used in an `rpc` definition.

Since a notification cannot be defined in a case statement, it is an error if a grouping that contains a notification at the top of its node hierarchy is used in a case definition.

If a leaf in the notification tree has a "mandatory" statement with the value "true", the leaf **MUST** be present in a notification instance.

If a leaf in the notification tree has a default value, the 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 notification instance with the default value as its value.

If a leaf-list in the notification tree has one or more default values, the client **MUST** use these values in the same cases as described in Section 7.7.2. In these cases, the client **MUST** operationally behave as if the leaf-list was present in the notification instance with the default values as its values.

If a "config" statement is present for any node in the notification tree, the "config" statement is ignored.

7.16.1. The notification's Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.16.2. NETCONF XML Encoding Rules

A notification node that is defined on the top-level of a module 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).

When a notification node is defined as a child to a data node, the <notification> element defined in NETCONF Event Notifications [RFC5277] contains an hierarchy of nodes that identifies the node in the datastore. It **MUST** contain all containers and list nodes from the top level down to the list or container containing the

notification. For lists, all key leafs MUST also be included. The last container or list contains an XML element that carries the name of the defined notification.

The notification's child nodes are encoded as subelements to the notification node's XML element, in any order.

7.16.3. Usage Example

The following example defines a notification at the top-level of a module:

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

  notification event {
    leaf event-class {
      type string;
    }
    leaf reporting-entity {
      type instance-identifier;
    }
    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>
      /ex:interface[ex:name='Ethernet0']
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

The following example defines a notification in a data node:

```
module interface-module {  
  yang-version 1.1;  
  namespace "http://example.com/schema/interfaces";  
  prefix "if";  
  
  container interfaces {  
    list interface {  
      key "name";  
      leaf name {  
        type string;  
      }  
      notification interface-enabled {  
        leaf by-user {  
          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>  
  <interfaces xmlns="http://example.com/schema/interfaces">  
    <interface>  
      <name>eth1</name>  
      <interface-enabled>  
        <by-user>fred</by-user>  
      </interface-enabled>  
    </interface>  
  </interfaces>  
</notification>
```

7.17. 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 14) 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 14) 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).

The "augment" statement MUST NOT add multiple nodes with the same name from the same module to the target node.

7.17.1. The augment's Substatements

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
uses	7.13	0..n
when	7.21.5	0..1

7.17.2. XML Encoding 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.

7.17.3. 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>
```

7.18. The identity Statement

The "identity" statement is used to define a new globally unique, abstract, and untyped identity. Its only purpose is to denote its name, semantics, and existence. An identity can either be defined from scratch or derived from one or more base identities. The identity's argument is an identifier that is the name of the identity. It is followed by a block of substatements that holds detailed identity information.

The built-in datatype "identityref" (see Section 9.10) can be used to reference identities within a data model.

7.18.1. The identity's Substatements

substatement	section	cardinality
base	7.18.2	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1

7.18.2. The base Statement

The "base" statement, which is optional, takes as an argument a string that is the name of an existing identity, from which the new identity is derived. If no "base" statement is present, the identity is defined from scratch. If multiple "base" statements are present, the identity is derived from all of them.

If a prefix is present on the base name, it refers to an identity defined in the module that was imported with that prefix, or the local module if the prefix matches the local module's prefix. Otherwise, an identity with the matching name **MUST** be defined in the current module or an included submodule.

An identity **MUST NOT** reference itself, neither directly nor indirectly through a chain of other identities.

The derivation of identities has the following properties:

- 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.

7.18.3. 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";
  }
}
```

7.19. 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 14.

An extension can allow refinement (see Section 7.13.2) and deviations (Section 7.20.3.2), but the mechanism for how this is defined is outside the scope of this specification.

7.19.1. The extension's Substatements

substatement	section	cardinality
argument	7.19.2	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1

7.19.2. 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.

7.19.2.1. The argument's Substatements

substatement	section	cardinality
yin-element	7.19.2.2	0..1

7.19.2.2. 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 13).

If no "yin-element" statement is present, it defaults to "false".

7.19.3. 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";  
  }  
}
```

7.20. Conformance-Related Statements

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

7.20.1. 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 7.20.2). Schema nodes tagged with an "if-feature" statement are ignored by the server unless the server supports the given feature expression. This allows portions of the YANG module to be conditional based on conditions in the server. The model can represent the abilities of the server within the model, giving a richer model that allows for differing server 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 server 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 server 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 server 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 server 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 server to implement a feature that is dependent on any other features (i.e., the feature has one or more "if-feature" substatements), the server MUST also implement all the dependant features.

7.20.1.1. The feature's Substatements

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
status	7.21.2	0..1
reference	7.21.4	0..1

7.20.2. The if-feature Statement

The "if-feature" statement makes its parent statement conditional. The argument is a boolean expression over feature names. In this expression, a feature name evaluates to "true" if and only if the feature is implemented by the server. The parent statement is implemented by servers where the boolean expression evaluates to "true".

The if-feature boolean expression syntax is formally defined by the rule "if-feature-expr" in Section 14. When this boolean expression is evaluated, the operator order of precedence is (highest precedence first): "not", "and", "or".

If a prefix is present on a feature name in the boolean expression, the prefixed name refers to a feature defined in the module that was imported with that prefix, or the local module if the prefix matches the local module's prefix. Otherwise, a feature with the matching name MUST be defined in the current module or an included submodule.

A leaf that is a list key MUST NOT have any "if-feature" statements.

7.20.2.1. Usage Example

In this example, the container "target" is implemented if any of the features "outbound-tls" or "outbound-ssh" is implemented by the server.

```
container target {  
  if-feature "outbound-tls or outbound-ssh";  
  ...  
}
```

7.20.3. The deviation Statement

The "deviation" statement defines a hierarchy of a module that the server 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 server or class of servers 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.

Server deviations are strongly discouraged and **MUST** only be used as a last resort. Telling the application how a server fails to follow a standard is no substitute for implementing the standard correctly. A server 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 server 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 servers to document portions of a base module that are not supported or supported but with different syntax, by using the "deviation" statement.

7.20.3.1. The deviation's Substatements

substatement	section	cardinality
description	7.21.3	0..1
deviate	7.20.3.2	1..n
reference	7.21.4	0..1

7.20.3.2. The deviate Statement

The "deviate" statement defines how the server'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 server.

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	7.21.1	0..1
default	7.6.4 7.7.4	0..n
mandatory	7.6.5	0..1
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
type	7.4	0..1
unique	7.8.3	0..n
units	7.3.3	0..1

The deviate's Substatements

If the target node has a property defined by an extension, this property can be deviated if the extension allows deviations. See Section 7.19 for details.

7.20.3.3. Usage Example

In this example, the server 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 server-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 server 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 server might remove this must constraint by doing:

```
    deviation "/base:system" {  
        deviate delete {  
            must "daytime or time";  
        }  
    }
```

7.21. Common Statements

This section defines substatements common to several other statements.

7.21.1. 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 are part of configuration datastores.

If "config" is "false", the definition represents state data. Data nodes representing state data are not part of configuration datastores.

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".

7.21.2. 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
}
```

7.21.3. 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.

7.21.4. 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";  
    ...  
}
```

7.21.5. The when Statement

The "when" statement makes its parent data definition statement conditional. The node defined by the parent data definition statement is only valid when the condition specified by the "when" statement is satisfied. The statement's argument is an XPath expression (see Section 6.4), which is used to formally specify this condition. If the XPath expression conceptually evaluates to "true" for a particular instance, then the node defined by the parent data definition statement is valid; otherwise, it is not.

A leaf that is a list key MUST NOT have a "when" statement.

See Section 8.2.2 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. If no such node exists, the context node is the root node. The accessible tree is tentatively altered during the processing of the XPath expression by removing all instances (if any) of the nodes added by the "augment" statement.
- 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. If no such node exists, the context node is the root node. The accessible tree is tentatively altered during the processing of the XPath expression by removing all instances (if any) of the nodes added by the "uses", "choice", or "case" statement.
- o If the "when" statement is a child of any other data definition statement, the accessible tree is tentatively altered during the processing of the XPath expression by replacing all instances of the data node for which the "when" statement is defined with a single dummy node with the same name, but with no value and no

children. If no such instance exists, the dummy node is tentatively created. The context node is this dummy node.

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

If the XPath expression references any node that also has associated "when" statements, these "when" expressions MUST be evaluated first. There MUST NOT be any circular dependencies in these "when" expressions.

Note that the XPath expression is conceptually evaluated. This means that an implementation does not have to use an XPath evaluator in the server. 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 valid state data tree.
- o If the constraint is defined on notification content, it MUST be true in any notification data tree.
- o If the constraint is defined on RPC or action input parameters, it MUST be true in an invocation of the RPC or action operation.
- o If the constraint is defined on RPC or action output parameters, it MUST be true in the RPC or action reply.

The following properties are true in all data trees:

- o All leaf data values MUST match the type constraints for the leaf, including those defined in the type's "range", "length", and "pattern" properties.
- o All key leafs MUST be present for all list entries.
- o Nodes MUST be present for at most one case branch in all choices.

- o There MUST be no nodes tagged with "if-feature" present if the "if-feature" expression evaluates to "false" in the server.
- o There MUST be no nodes tagged with "when" present if the "when" condition evaluates to "false" in the data tree.

The following properties are true in a valid data tree:

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

The running configuration datastore MUST always be valid.

8.2. NETCONF 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.

8.2.1. 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 server implements.

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

- o If data for more than one case branch of a choice is present, the server MUST reply with a "bad-element" in the rpc-error.
- o If data for a node tagged with "if-feature" is present, and the if-feature expression evaluates to "false" in the server, 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.

8.2.2. 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.
- o Modification requests for nodes tagged with "when", and the "when" condition evaluates to "false". In this case the server MUST reply with an "unknown-element" error-tag in the rpc-error.

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.

8.2.3. 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.

9. Built-In Types

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

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

The different built-in types and their derived types allow different kinds of subtyping, namely length and regular expression restrictions of strings (Section 9.4.4, Section 9.4.5) and range restrictions of numeric types (Section 9.2.4).

The lexical representation of a value of a certain type is used in the XML encoding 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 server sends data encoded in XML, it MUST use the canonical form defined in this section. Other encodings may introduce alternate representations. Note, however, that values in the data tree are conceptually stored in the canonical representation as defined in this section. In particular, any XPath expression evaluations are done using the canonical form.

Some types have a lexical representation that depends on the encoding, e.g., 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 encoding, 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 14.

9.2.4.1. The range's Substatements

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	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 the Unicode and ISO/IEC 10646 [ISO.10646] characters, including tab, carriage return, and line feed but excluding the other C0 control characters, the surrogate blocks, and the noncharacters. The string syntax is formally defined by the rule "yang-string" in Section 14.

9.4.1. Lexical Representation

A string value is lexically represented as character data in the XML encoding.

9.4.2. Canonical Form

The canonical form is the same as the lexical representation. No Unicode normalization is performed of string values.

9.4.3. Restrictions

A string can be restricted with the "length" (Section 9.4.4) and "pattern" (Section 9.4.5) statements.

9.4.4. The length Statement

The "length" statement, which is an optional substatement to the "type" statement, takes as an argument a length expression string. It is used to restrict the built-in types "string" and "binary" or types derived from them.

A "length" statement restricts the number of Unicode characters in the string.

A length range consists of an explicit value, or a lower bound, two consecutive dots "..", and an upper bound. Multiple values or ranges can be given, separated by "|". Length-restricting values MUST NOT be negative. If multiple values or ranges are given, they all MUST be disjoint and MUST be in ascending order. If a length restriction

is applied to an already length-restricted type, the new restriction MUST be equal or more limiting, that is, raising the lower bounds, reducing the upper bounds, removing explicit length values or ranges, or splitting ranges into multiple ranges with intermediate gaps. A length value is a non-negative integer, or one of the special values "min" or "max". "min" and "max" mean the minimum and maximum length accepted for the type being restricted, respectively. An implementation is not required to support a length value larger than 18446744073709551615.

The length expression syntax is formally defined by the rule "length-arg" in Section 14.

9.4.4.1. The length's Substatements

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

9.4.5. The pattern Statement

The "pattern" statement, which is an optional substatement to the "type" statement, takes as an argument a regular expression string, as defined in [XSD-TYPES]. It is used to restrict the built-in type "string", or types derived from "string", to values that match the pattern.

If the type has multiple "pattern" statements, the expressions are ANDed together, i.e., all such expressions have to match.

If a pattern restriction is applied to an already pattern-restricted type, values must match all patterns in the base type, in addition to the new patterns.

9.4.5.1. The pattern's Substatements

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
modifier	9.4.6	0..1
reference	7.21.4	0..1

9.4.6. The modifier Statement

The "modifier" statement, which is an optional substatement to the "pattern" statement, takes as an argument the string "invert-match".

If a pattern has the "invert-match" modifier present, the type is restricted to values that do not match the pattern.

9.4.7. Usage Example

With the following typedef:

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

the following refinement is legal:

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

and the following refinement is illegal:

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

With the following type:

```
type string {
    length "0..4";
    pattern "[0-9a-fA-F]*";
}
```

the following strings match:

```
AB          // legal
9A00        // legal
```

and the following strings do not match:

```
00ABAB      // illegal, too long
xx00        // illegal, bad characters
```

With the following type:

```
typedef yang-identifier {
  type string {
    length "1..max";
    pattern '[a-zA-Z_][a-zA-Z0-9\-\_\.]*';
    pattern '[xX][mM][lL].*' {
      modifier invert-match;
    }
  }
}
```

the following string match:

```
enabled      // legal
```

and the following strings do not match:

```
10-mbit      // illegal, starts with a number
xml-element  // illegal, starts with illegal sequence
```

9.5. The boolean Built-In Type

The boolean built-in type represents a boolean value.

9.5.1. Lexical Representation

The lexical representation of a boolean value is a string with a value of "true" or "false". These values MUST be in lowercase.

9.5.2. Canonical Form

The canonical form is the same as the lexical representation.

9.5.3. Restrictions

A boolean cannot be restricted.

9.6. The enumeration Built-In Type

The enumeration built-in type represents values from a set of assigned names.

9.6.1. Lexical Representation

The lexical representation of an enumeration value is the assigned name string.

9.6.2. Canonical Form

The canonical form is the assigned name string.

9.6.3. Restrictions

An enumeration can be restricted with the "enum" (Section 9.6.4) statement.

9.6.4. The enum Statement

The "enum" statement, which is a substatement to the "type" statement, MUST be present if the type is "enumeration". It is repeatedly used to specify each assigned name of an enumeration type. It takes as an argument a string which is the assigned name. The string MUST NOT be empty and MUST NOT have any leading or trailing whitespace characters. The use of Unicode control codes SHOULD be avoided.

The statement is optionally followed by a block of substatements that holds detailed enum information.

All assigned names in an enumeration MUST be unique.

When an existing enumeration type is restricted, the set of assigned names in the new type MUST be a subset of the base type's set of assigned names. The value of such an assigned name MUST not be changed.

9.6.4.1. The enum's Substatements

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
value	9.6.4.2	0..1

9.6.4.2. The value Statement

The "value" statement, which is optional, is used to associate an integer value with the assigned name for the enum. This integer value MUST be in the range -2147483648 to 2147483647, and it MUST be unique within the enumeration type.

If a value is not specified, then one will be automatically assigned. If the "enum" substatement is the first one defined, the assigned value is zero (0); otherwise, the assigned value is one greater than the current highest enum value (i.e., the highest enum value, implicit or explicit, prior to the current "enum" substatement in the parent "type" statement).

If the current highest value is equal to 2147483647, then an enum value MUST be specified for "enum" substatements following the one with the current highest value.

When an existing enumeration type is restricted, the "value" statement MUST either have the same value as in the base type or not be present, in which case the value is the same as in the base type.

9.6.5. Usage Example

```
leaf myenum {
  type enumeration {
    enum zero;
    enum one;
    enum seven {
      value 7;
    }
  }
}
```

The lexical representation of the leaf "myenum" with value "seven" is:

```
<myenum>seven</myenum>
```

With the following typedef:

```
typedef my-base-enumeration-type {  
  type enumeration {  
    enum white {  
      value 1;  
    }  
    enum yellow {  
      value 2;  
    }  
    enum red {  
      value 3;  
    }  
  }  
}
```

the following refinement is legal:

```
type my-base-enumeration-type {  
  // legal enum refinement  
  enum yellow;  
  enum red {  
    value 3;  
  }  
}
```

and the following refinement is illegal:

```
type my-base-enumeration-type {  
  // illegal enum refinement  
  enum yellow {  
    value 4; // illegal value change  
  }  
  enum black; // illegal addition of new name  
}
```

9.7. The bits Built-In Type

The bits built-in type represents a bit set. That is, a bits value is a set of flags identified by small integer position numbers starting at 0. Each bit number has an assigned name.

9.7.1. Restrictions

A bits type cannot be restricted.

9.7.2. Lexical Representation

The lexical representation of the bits type is a space-separated list of the individual bit values that are set. An empty string thus represents a value where no bits are set.

9.7.3. Canonical Form

In the canonical form, the bit values are separated by a single space character and they appear ordered by their position (see Section 9.7.4.2).

9.7.4. The bit Statement

The "bit" statement, which is a substatement to the "type" statement, MUST be present if the type is "bits". It is repeatedly used to specify each assigned named bit of a bits type. It takes as an argument a string that is the assigned name of the bit. It is followed by a block of substatements that holds detailed bit information. The assigned name follows the same syntax rules as an identifier (see Section 6.2).

All assigned names in a bits type MUST be unique.

9.7.4.1. The bit's Substatements

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	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.

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 type can be restricted with the "length" (Section 9.4.4) statement. The length of a binary value is the number of octets it contains.

9.8.2. Lexical Representation

Binary values are encoded with the base64 encoding scheme (see [RFC4648], Section 4).

9.8.3. Canonical Form

The canonical form of a binary value follows the rules in [RFC4648].

9.9. The leafref Built-In Type

The leafref type is used to declare a constraint on the value space of a leaf, based on a reference to a set of leaf instances in the data tree. The "path" substatement (Section 9.9.2) selects a set of leaf instances, and the leafref value space is the set of values of these leaf instances.

If the leaf with the leafref type represents configuration data, and the "require-instance" property (Section 9.9.3) is "true", the leaf it refers to MUST also represent configuration. Such a leaf puts a constraint on valid data. All such nodes MUST reference existing leaf instances or leaves with default values in use (see Section 7.6.1 and Section 7.7.2) for the data to be valid. This constraint is enforced according to the rules in Section 8.

There MUST NOT be any circular chains of leafrefs.

If the leaf that the leafref refers to is conditional based on one or more features (see Section 7.20.2), then the leaf with the leafref type MUST also be conditional based on at least the same set of features.

9.9.1. Restrictions

A leafref can be restricted with the "require-instance" statement (Section 9.9.3).

9.9.2. The path Statement

The "path" statement, which is a substatement to the "type" statement, MUST be present if the type is "leafref". It takes as an argument a string that MUST refer to a leaf or leaf-list node.

The syntax for a path argument is a subset of the XPath abbreviated syntax. Predicates are used only for constraining the values for the key nodes for list entries. Each predicate consists of exactly one equality test per key, and multiple adjacent predicates MAY be present if a list has multiple keys. The syntax is formally defined by the rule "path-arg" in Section 14.

The predicates are only used when more than one key reference is needed to uniquely identify a leaf instance. This occurs if a list has multiple keys, or a reference to a leaf other than the key in a

list is needed. In these cases, multiple leafrefs are typically specified, and predicates are used to tie them together.

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

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

- o If the "path" statement is defined within a typedef, the context node is the leaf or leaf-list node in the data tree that references the typedef.
- o Otherwise, the context node is the node in the data tree for which the "path" statement is defined.

9.9.3. The require-instance Statement

The "require-instance" statement, which is a substatement to the "type" statement, **MAY** be present if the type is "instance-identifier" or "leafref". It takes as an argument the string "true" or "false". If this statement is not present, it defaults to "true".

If "require-instance" is "true", it means that the instance being referred **MUST** exist for the data to be valid. This constraint is enforced according to the rules in Section 8.

If "require-instance" is "false", it means that the instance being referred **MAY** exist in valid data.

9.9.4. Lexical Representation

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

9.9.5. Canonical Form

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

9.9.6. Usage Example

With the following list:

```
list interface {  
  key "name";  
  leaf name {  
    type string;  
  }  
  leaf admin-status {  
    type admin-status;  
  }  
  list address {  
    key "ip";  
    leaf ip {  
      type yang:ip-address;  
    }  
  }  
}
```

The following leafref refers to an existing interface:

```
leaf mgmt-interface {  
  type leafref {  
    path "../interface/name";  
  }  
}
```

An example of a corresponding XML snippet:

```
<interface>  
  <name>eth0</name>  
</interface>  
<interface>  
  <name>lo</name>  
</interface>  
  
<mgmt-interface>eth0</mgmt-interface>
```

The following leafrefs refer to an existing address of an interface:

```
container default-address {  
  leaf ifname {  
    type leafref {  
      path "../../interface/name";  
    }  
  }  
  leaf address {  
    type leafref {  
      path "../../interface[name = current()../ifname]"  
        + "/address/ip";  
    }  
  }  
}
```

An example of a corresponding XML snippet:

```
<interface>  
  <name>eth0</name>  
  <admin-status>up</admin-status>  
  <address>  
    <ip>192.0.2.1</ip>  
  </address>  
  <address>  
    <ip>192.0.2.2</ip>  
  </address>  
</interface>  
<interface>  
  <name>lo</name>  
  <admin-status>up</admin-status>  
  <address>  
    <ip>127.0.0.1</ip>  
  </address>  
</interface>  
  
<default-address>  
  <ifname>eth0</ifname>  
  <address>192.0.2.2</address>  
</default-address>
```

The following list uses a leafref for one of its keys. This is similar to a foreign key in a relational database.

```
list packet-filter {  
  key "if-name filter-id";  
  leaf if-name {  
    type leafref {  
      path "/interface/name";  
    }  
  }  
  leaf filter-id {  
    type uint32;  
  }  
  ...  
}
```

An example of a corresponding XML snippet:

```
<interface>  
  <name>eth0</name>  
  <admin-status>up</admin-status>  
  <address>  
    <ip>192.0.2.1</ip>  
  </address>  
  <address>  
    <ip>192.0.2.2</ip>  
  </address>  
</interface>  
  
<packet-filter>  
  <if-name>eth0</if-name>  
  <filter-id>1</filter-id>  
  ...  
</packet-filter>  
<packet-filter>  
  <if-name>eth0</if-name>  
  <filter-id>2</filter-id>  
  ...  
</packet-filter>
```

The following notification defines two leafrefs to refer to an existing admin-status:

```
notification link-failure {  
  leaf if-name {  
    type leafref {  
      path "/interface/name";  
    }  
  }  
  leaf admin-status {  
    type leafref {  
      path "/interface[name = current()../if-name]"  
        + "/admin-status";  
    }  
  }  
}
```

An example of a corresponding XML notification:

```
<notification  
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">  
  <eventTime>2008-04-01T00:01:00Z</eventTime>  
  <link-failure xmlns="http://acme.example.com/system">  
    <if-name>eth0</if-name>  
    <admin-status>up</admin-status>  
  </link-failure>  
</notification>
```

9.10. The identityref Built-In Type

The identityref type is used to reference an existing identity (see Section 7.18).

9.10.1. Restrictions

An identityref cannot be restricted.

9.10.2. The identityref's base Statement

The "base" statement, which is a substatement to the "type" statement, MUST be present at least once if the type is "identityref". The argument is the name of an identity, as defined by an "identity" statement. If a prefix is present on the identity name, it refers to an identity defined in the module that was imported with that prefix. Otherwise, an identity with the matching name MUST be defined in the current module or an included submodule.

Valid values for an identityref are any identities derived from all the identityref's base identities. On a particular server, the valid values are further restricted to the set of identities defined in the modules implemented by the server.

9.10.3. Lexical Representation

An identityref is lexically represented as the referred identity's qualified name as defined in [XML-NAMES]. If the prefix is not present, the namespace of the identityref is the default namespace in effect on the element that contains the identityref value.

When an identityref is given a default value using the "default" statement, the identity name in the default value MAY have a prefix. If a prefix is present on the identity name, it refers to an identity defined in the module that was imported with that prefix, or the prefix for the current module if the identity is defined in the current module or one of its submodules. Otherwise, an identity with the matching name MUST be defined in the current module or one of its submodules.

The string value of a node of type identityref in a "must" or "when" XPath expression is the referred identity's qualified name with the prefix present. If the referred identity is defined in an imported module, the prefix in the string value is the prefix defined in the corresponding "import" statement. Otherwise, the prefix in the string value is the prefix for the current module.

9.10.4. Canonical Form

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

9.10.5. Usage Example

With the identity definitions in Section 7.18.3 and the following module:

```
module my-crypto {  
  yang-version 1.1;  
  namespace "http://example.com/my-crypto";  
  prefix mc;  
  
  import "crypto-base" {  
    prefix "crypto";  
  }  
  
  identity aes {  
    base "crypto:crypto-alg";  
  }  
  
  leaf crypto {  
    type identityref {  
      base "crypto:crypto-alg";  
    }  
  }  
  
  container aes-parameters {  
    when "../crypto = 'mc:aes'";  
    ...  
  }  
}
```

the following is an example how the leaf "crypto" can be encoded, if the value is the "des3" identity defined in the "des" module:

```
<crypto xmlns:des="http://example.com/des">des:des3</crypto>
```

Any prefixes used in the encoding are local to each instance encoding. This means that the same identityref may be encoded differently by different implementations. For example, the following example encodes the same leaf as above:

```
<crypto xmlns:x="http://example.com/des">x:des3</crypto>
```

If the "crypto" leaf's value instead is "aes" defined in the "my-crypto" module, it can be encoded as:

```
<crypto xmlns:mc="http://example.com/my-crypto">mc:aes</crypto>
```

or, using the default namespace:

```
<crypto>aes</crypto>
```

9.11. The empty Built-In Type

The empty built-in type represents a leaf that does not have any value, it conveys information by its presence or absence.

An empty type cannot have a default value.

9.11.1. Restrictions

An empty type cannot be restricted.

9.11.2. Lexical Representation

Not applicable.

9.11.3. Canonical Form

Not applicable.

9.11.4. Usage Example

With the following leaf

```
leaf enable-qos {  
  type empty;  
}
```

the following is an example of a valid encoding

```
<enable-qos/>
```

if the leaf exists.

9.12. The union Built-In Type

The union built-in type represents a value that corresponds to one of its member types.

When the type is "union", the "type" statement (Section 7.4) MUST be present. It is used to repeatedly specify each member type of the union. It takes as an argument a string that is the name of a member type.

A member type can be of any built-in or derived type.

In the XML encoding, a value representing a union data type is validated consecutively against each member type, in the order they are specified in the "type" statement, until a match is found. The

type that matched will be the type of the value for the node that was validated.

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

9.12.1. Restrictions

A union cannot be restricted. However, each member type can be restricted, based on the rules defined in Section 9.

9.12.2. Lexical Representation

The lexical representation of a union is a value that corresponds to the representation of any one of the member types.

9.12.3. Canonical Form

The canonical form of a union value is the same as the canonical form of the member type of the value.

9.12.4. Usage Example

The following is a union of an int32 and an enumeration:

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

Care must be taken when a member type is a leafref where the "require-instance" property (Section 9.9.3) is "true". If a leaf of such a type refers to an existing instance, the leaf's value must be re-validated if the target instance is deleted. For example, with the following definitions:

```
list filter {
  key name;
  leaf name {
    type string;
  }
  ...
}

leaf outbound-filter {
  type union {
    type leafref {
      path "/filter/name";
    }
    type enumeration {
      enum default-filter;
    }
  }
}
```

assume that there exists an entry in the filter list with the name "http", and that the outbound-filter leaf has this value:

```
<filter>
  <name>http</name>
</filter>
<outbound-filter>http</outbound-filter>
```

If the filter entry "http" is removed, the outbound-filter leaf's value doesn't match the leafref, and the next member type is checked. The current value ("http") doesn't match the enumeration, so the resulting configuration is invalid.

If the second member type in the union had been of type "string" instead of an enumeration, the current value would have matched, and the resulting configuration would have been valid.

9.13. The instance-identifier Built-In Type

The instance-identifier built-in type is used to uniquely identify a particular instance node in the data tree.

The syntax for an instance-identifier is a subset of the XPath abbreviated syntax, formally defined by the rule "instance-identifier" in Section 14. It is used to uniquely identify a node in the data tree. Predicates are used only for specifying the values for the key nodes for list entries, a value of a leaf-list entry, or a positional index for a list without keys. For identifying list entries with keys, each predicate consists of one

equality test per key, and each key MUST have a corresponding predicate.

If the leaf with the instance-identifier type represents configuration data, and the "require-instance" property (Section 9.9.3) is "true", the node it refers to MUST also represent configuration. Such a leaf puts a constraint on valid data. All such leaf nodes MUST reference existing nodes or leaf or leaf-list nodes with their default value in use (see Section 7.6.1 and Section 7.7.2) for the data to be valid. This constraint is enforced according to the rules in Section 8.

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

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

9.13.1. Restrictions

An instance-identifier can be restricted with the "require-instance" statement (Section 9.9.3).

9.13.2. Lexical Representation

An instance-identifier value is lexically represented as a string. All node names in an instance-identifier value MUST be qualified with explicit namespace prefixes, and these prefixes MUST be declared in the XML namespace scope in the instance-identifier's XML element.

Any prefixes used in the encoding are local to each instance encoding. This means that the same instance-identifier may be encoded differently by different implementations.

9.13.3. Canonical Form

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

9.13.4. Usage Example

The following are examples of instance identifiers:

```
/* instance-identifier for a container */
/ex:system/ex:services/ex:ssh

/* instance-identifier for a leaf */
/ex:system/ex:services/ex:ssh/ex:port

/* instance-identifier for a list entry */
/ex:system/ex:user[ex:name='fred']

/* instance-identifier for a leaf in a list entry */
/ex:system/ex:user[ex:name='fred']/ex:type

/* instance-identifier for a list entry with two keys */
/ex:system/ex:server[ex:ip='192.0.2.1'][ex:port='80']

/* instance-identifier for a leaf-list entry */
/ex:system/ex:services/ex:ssh/ex:cipher[.='blowfish-cbc']

/* instance-identifier for a list entry without keys */
/ex:stats/ex:port[3]
```

10. XPath Functions

This document defines two generic XPath functions and four YANG type-specific XPath functions. The function signatures are specified with the syntax used in [XPATH].

10.1. Functions for Node Sets

10.1.1. current()

node-set current()

The function current() takes no input parameters, and returns a node set with the initial context node as its only member.

10.2. Functions for Strings

10.2.1. re-match()

boolean re-match(string subject, string pattern)

The re-match() function returns true if the "subject" string matches the regular expression "pattern"; otherwise it returns false.

The function "re-match" checks if a string matches a given regular expression. The regular expressions used are the XML Schema regular

expressions [XSD-TYPES]. Note that this includes implicit anchoring of the regular expression at the head and tail.

10.2.1.1. Usage Example

The expression:

```
re-match('1.22.333', '\d{1,3}\.\d{1,3}\.\d{1,3}')
```

returns true.

To count all logical interfaces called eth0.<number>, do:

```
count(/interface[re-match(name, 'eth0\.\d+')])
```

10.3. Functions for the YANG Types "leafref" and "instance-identifier"

10.3.1. deref()

```
node-set deref(node-set nodes)
```

The deref() function follows the reference defined by the first node in document order in the argument "nodes", and returns the nodes it refers to.

If the first argument node is of type instance-identifier, the function returns a node set that contains the single node that the instance identifier refers to, if it exists. If no such node exists, an empty node-set is returned.

If the first argument node is of type leafref, the function returns a node set that contains the nodes that the leafref refers to.

If the first argument node is of any other type, an empty node set is returned.

10.3.1.1. Usage Example


```
list interface {
  key "name type";
  leaf name { ... }
  leaf type { ... }
  leaf enabled {
    type boolean;
  }
  ...
}

container mgmt-interface {
  leaf name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf type {
    type leafref {
      path "/interface[name=current()/../name]/type";
      must 'deref(..)/../enabled = "true"' {
        error-message
          "The management interface cannot be disabled.";
      }
    }
  }
}
```

10.4. Functions for the YANG Type "identityref"

10.4.1. derived-from()

```
boolean derived-from(node-set nodes,
                     string module-name,
                     string identity-name)
```

The `derived-from()` function returns true if the first node in document order in the argument "nodes" is a node of type `identityref`, and its value is an identity that is derived from the identity "identity-name" defined in the YANG module "module-name"; otherwise it returns false.

10.4.2. derived-from-or-self()

```
boolean derived-from-or-self(node-set nodes,
                             string module-name,
                             string identity-name)
```

The `derived-from-or-self()` function returns true if the first node in document order in the argument "nodes" is a node of type `identityref`,

and its value is an identity that is equal to or derived from the identity "identity-name" defined in the YANG module "module-name"; otherwise it returns false.

10.4.2.1. Usage Example

```
module example-interface {
  ...

  identity interface-type;

  identity ethernet {
    base interface-type;
  }

  identity fast-ethernet {
    base ethernet;
  }

  identity gigabit-ethernet {
    base ethernet;
  }

  list interface {
    key name;
    ...
    leaf type {
      type identityref {
        base interface-type;
      }
    }
    ...
  }

  augment "/interface" {
    when 'derived-from(type,
                        "example-interface",
                        "ethernet")';
    // ethernet-specific definitions here
  }
}
```

10.5. Functions for the YANG Type "enumeration"

10.5.1. enum-value()

```
number enum-value(node-set nodes)
```

The enum-value() function checks if the first node in document order in the argument "nodes" is a node of type enumeration, and returns the enum's integer value. If the "nodes" node set is empty, or if the first node in "nodes" is not of type enumeration, it returns NaN.

10.5.1.1. Usage Example

With this data model:

```
list alarm {  
  ...  
  leaf severity {  
    type enumeration {  
      enum cleared {  
        value 1;  
      }  
      enum indeterminate {  
        value 2;  
      }  
      enum minor {  
        value 3;  
      }  
      enum warning {  
        value 4;  
      }  
      enum major {  
        value 5;  
      }  
      enum critical {  
        value 6;  
      }  
    }  
  }  
}
```

the following XPath expression selects only alarms that are of severity "major" or higher:

```
/alarm[enum-value(severity) >= 5]
```

10.6. Functions for the YANG Type "bits"

10.6.1. bit-is-set()

```
boolean bit-is-set(node-set nodes, string bit-name)
```

The `bit-is-set()` function returns true if the first node in document order in the argument "nodes" is a node of type bits, and its value has the bit "'bit-name" set; otherwise it returns false.

10.6.1.1. Usage Example

If an interface has this leaf:

```
leaf flags {  
  type bits {  
    bit UP;  
    bit PROMISCUOUS  
    bit DISABLED;  
  }  
}
```

the following XPath expression can be used to select all interfaces with the UP flag set:

```
/interface[bit-is-set(flags, "UP")]
```

11. Updating a Module

As experience is gained with a module, it may be desirable to revise that module. However, changes are not allowed if they have any potential to cause interoperability problems between a client using an original specification and a server using an updated specification.

For any published change, a new "revision" statement (Section 7.1.9) MUST be included in front of the existing "revision" statements. If there are no existing "revision" statements, then one MUST be added to identify the new revision. Furthermore, any necessary changes MUST be applied to any meta-data statements, including the "organization" and "contact" statements (Section 7.1.7, Section 7.1.8).

Note that definitions contained in a module are available to be imported by any other module, and are referenced in "import" statements via the module name. Thus, a module name MUST NOT be changed. Furthermore, the "namespace" statement MUST NOT be changed, since all XML elements are qualified by the namespace.

Obsolete definitions MUST NOT be removed from modules since their identifiers may still be referenced by other modules.

A definition may be revised in any of the following ways:

- o An "enumeration" type may have new enums added, provided the old enums's values do not change. Note that inserting a new enum before an existing enum or reordering existing enums will result in new values for the existing enums, unless they have explicit values assigned to them.
- o A "bits" type may have new bits added, provided the old bit positions do not change. Note that inserting a new bit before an existing bit or reordering existing bit will result in new positions for the existing bits, unless they have explicit positions assigned to them.
- o A "range", "length", or "pattern" statement may expand the allowed value space.
- o A "default" statement may be added to a leaf that does not have a default value (either directly or indirectly through its type).
- o A "units" statement may be added.
- o A "reference" statement may be added or updated.
- o A "must" statement may be removed or its constraint relaxed.
- o A "mandatory" statement may be removed or changed from "true" to "false".
- o A "min-elements" statement may be removed, or changed to require fewer elements.
- o A "max-elements" statement may be removed, or changed to allow more elements.
- o A "description" statement may be added or clarified without changing the semantics of the definition.
- o A "base" statement may be added to an "identity" statement.
- o A "base" statement may be removed from an "identityref" type, provided there is at least one "base" statement left.
- o New typedefs, groupings, rpcs, notifications, extensions, features, and identities may be added.

- o New data definition statements may be added if they do not add mandatory nodes (Section 3) 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).
- o An "if-feature" statement may be removed, provided its node is not mandatory (Section 3).
- o A "status" statement may be added, or changed from "current" to "deprecated" or "obsolete", or from "deprecated" to "obsolete".
- o A "type" statement may be replaced with another "type" statement that does not change the syntax or semantics of the type. For example, an inline type definition may be replaced with a typedef, but an int8 type cannot be replaced by an int16, since the syntax would change.
- o Any set of data definition nodes may be replaced with another set of syntactically and semantically equivalent nodes. For example, a set of leafs may be replaced by a uses of a grouping with the same leafs.
- o A module may be split into a set of submodules, or a submodule may be removed, provided the definitions in the module do not change in any other way than allowed here.
- o The "prefix" statement may be changed, provided all local uses of the prefix also are changed.

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

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

12. Coexistence with YANG version 1

A YANG version 1.1 module MUST NOT include a YANG version 1 submodule, and a YANG version 1 module MUST NOT include a YANG version 1.1 submodule.

A YANG version 1 module or submodule MUST NOT import a YANG version 1.1 module by revision.

A YANG version 1.1 module or submodule MAY import a YANG version 1 module by revision.

If a YANG version 1 module A imports module B without revision, and module B is updated to YANG version 1.1, a server MAY implement both these modules at the same time. In such cases, a NETCONF server MUST advertise both modules using the rules defined in Section 5.6.5, and SHOULD advertise module A and the latest revision of module B that is specified with YANG version 1 according to the rules defined in [RFC6020].

This rule exists in order to allow implementations of existing YANG version 1 modules together with YANG version 1.1 modules. Without this rule, updating a single module to YANG version 1.1 would have a cascading effect on modules that import it, requiring all of them to also be updated to YANG version 1.1, and so on.

13. 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.

13.1. 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 7.19). 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
action	name	false
anydata	name	false
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
modifier	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.

13.1.1.1. 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 7.19.3, 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>
```

14. YANG ABNF Grammar

In YANG, almost all statements are unordered. The ABNF grammar [RFC5234] [RFC7405] defines the canonical order. To improve module readability, it is RECOMMENDED that clauses be entered in this order.

Within the ABNF grammar, unordered statements are marked with comments.

This grammar assumes that the scanner replaces YANG comments with a single space character.

<CODE BEGINS> file "yang.abnf"

```
module-stmt      = optsep module-keyword sep identifier-arg-str
                  optsep
                  "{" stmtsep
                    module-header-stmts
                    linkage-stmts
                    meta-stmts
                    revision-stmts
```

```
        body-stmts
    "}" optsep

submodule-stmt    = optsep submodule-keyword sep identifier-arg-str
                  optsep
                  "{" stmtsep
                  submodule-header-stmts
                  linkage-stmts
                  meta-stmts
                  revision-stmts
                  body-stmts
                  "}" optsep

module-header-stmts = ;; these stmts can appear in any order
                      yang-version-stmt
                      namespace-stmt
                      prefix-stmt

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

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

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

revision-stmts    = *revision-stmt

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

data-def-stmt     = container-stmt /
                      leaf-stmt /
```

```
leaf-list-stmt /
list-stmt /
choice-stmt /
anydata-stmt /
anyxml-stmt /
uses-stmt

yang-version-stmt  = yang-version-keyword sep yang-version-arg-str
                    stmtend

yang-version-arg-str = < a string that matches the rule >
                      < yang-version-arg >

yang-version-arg    = "1.1"

import-stmt         = import-keyword sep identifier-arg-str optsep
                    "{" stmtsep
                      prefix-stmt
                      [revision-date-stmt]
                    "}" stmtsep

include-stmt        = include-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                       [revision-date-stmt]
                     "}") stmtsep

namespace-stmt      = namespace-keyword sep uri-str stmtend

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

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

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

organization-stmt   = organization-keyword sep string stmtend

contact-stmt        = contact-keyword sep string stmtend

description-stmt    = description-keyword sep string stmtend

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

```
units-stmt          = units-keyword sep string stmtend

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

revision-date       = date-arg-str

revision-date-stmt  = revision-date-keyword sep revision-date stmtend

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

argument-stmt       = argument-keyword sep identifier-arg-str optsep
                    ( ";" /
                      "{" stmtsep
                        [yin-element-stmt]
                      "}") stmtsep

yin-element-stmt    = yin-element-keyword sep yin-element-arg-str
                    stmtend

yin-element-arg-str = < a string that matches the rule >
                    < yin-element-arg >

yin-element-arg     = true-keyword / false-keyword

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

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

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

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

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

if-feature-expr    = "(" if-feature-expr ")" /
                    if-feature-expr sep boolean-operator sep
                    if-feature-expr /
                    not-keyword sep if-feature-expr /
                    identifier-ref-arg

boolean-operator   = and-keyword / or-keyword

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

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

type-body-stmts    = numerical-restrictions /
                    decimal64-specification /
                    string-restrictions /
                    enum-specification /
```

```
leafref-specification /
identityref-specification /
instance-identifier-specification /
bits-specification /
union-specification /
binary-specification

numerical-restrictions = range-stmt

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

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

fraction-digits-stmt = fraction-digits-keyword sep
                      fraction-digits-arg-str stmtend

fraction-digits-arg-str = < a string that matches the rule >
                        < fraction-digits-arg >

fraction-digits-arg = ("1" ["0" / "1" / "2" / "3" / "4" /
                             "5" / "6" / "7" / "8"])
                    / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

string-restrictions = ;; these stmts can appear in any order
                    [length-stmt]
                    *pattern-stmt

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

pattern-stmt          = pattern-keyword sep string optsep
```



```
( ";" /
  "{" stmtsep
    ;; these stmts can appear in any order
    [modifier-stmt]
    [error-message-stmt]
    [error-app-tag-stmt]
    [description-stmt]
    [reference-stmt]
  "}") stmtsep

modifier-stmt      = modifier-keyword sep modifier-arg-str stmtend

modifier-arg-str    = < a string that matches the rule >
                     < modifier-arg >

modifier-arg        = invert-match-keyword

default-stmt        = default-keyword sep string stmtend

enum-specification  = 1*enum-stmt

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

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

path-stmt           = path-keyword sep path-arg-str stmtend

require-instance-stmt = require-instance-keyword sep
                        require-instance-arg-str stmtend

require-instance-arg-str = < a string that matches the rule >
                           < require-instance-arg >

require-instance-arg = true-keyword / false-keyword
```

```
instance-identifier-specification =
    [require-instance-stmt]

identityref-specification =
    1*base-stmt

union-specification = 1*type-stmt

binary-specification = [length-stmt]

bits-specification  = 1*bit-stmt

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

position-stmt        = position-keyword sep
                      position-value-arg-str stmtend

position-value-arg-str = < a string that matches the rule >
                      < position-value-arg >

position-value-arg    = non-negative-integer-value

status-stmt           = status-keyword sep status-arg-str stmtend

status-arg-str        = < a string that matches the rule >
                      < status-arg >

status-arg            = current-keyword /
                      obsolete-keyword /
                      deprecated-keyword

config-stmt           = config-keyword sep
                      config-arg-str stmtend

config-arg-str        = < a string that matches the rule >
                      < config-arg >

config-arg            = true-keyword / false-keyword
```

```
mandatory-stmt      = mandatory-keyword sep
                      mandatory-arg-str stmtend

mandatory-arg-str    = < a string that matches the rule >
                      < mandatory-arg >

mandatory-arg        = true-keyword / false-keyword

presence-stmt       = presence-keyword sep string stmtend

ordered-by-stmt      = ordered-by-keyword sep
                      ordered-by-arg-str stmtend

ordered-by-arg-str   = < a string that matches the rule >
                      < ordered-by-arg >

ordered-by-arg       = user-keyword / system-keyword

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

error-message-stmt   = error-message-keyword sep string stmtend

error-app-tag-stmt   = error-app-tag-keyword sep string stmtend

min-elements-stmt    = min-elements-keyword sep
                      min-value-arg-str stmtend

min-value-arg-str    = < a string that matches the rule >
                      < min-value-arg >

min-value-arg        = non-negative-integer-value

max-elements-stmt    = max-elements-keyword sep
                      max-value-arg-str stmtend

max-value-arg-str    = < a string that matches the rule >
                      < max-value-arg >

max-value-arg        = unbounded-keyword /
                      positive-integer-value
```

```
value-stmt          = value-keyword sep integer-value-str stmtend

integer-value-str    = < a string that matches the rule >
                      < integer-value >

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

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

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

```

        [description-stmt]
        [reference-stmt]
    "}" stmtsep

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

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

key-stmt            = key-keyword sep key-arg-str stmtend

key-arg-str         = < a string that matches the rule >
                    < key-arg >

```

```
key-arg          = node-identifier *(sep node-identifier)

unique-stmt      = unique-keyword sep unique-arg-str stmtend

unique-arg-str   = < a string that matches the rule >
                  < unique-arg >

unique-arg       = descendant-schema-nodeid
                  *(sep descendant-schema-nodeid)

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

short-case-stmt  = choice-stmt /
                  container-stmt /
                  leaf-stmt /
                  leaf-list-stmt /
                  list-stmt /
                  anydata-stmt /
                  anyxml-stmt

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

anydata-stmt     = anydata-keyword sep identifier-arg-str optsep
                  ( ";" /
                    "{" stmtsep
```

```
        ;; these stmts can appear in any order
        [when-stmt]
        *if-feature-stmt
        *must-stmt
        [config-stmt]
        [mandatory-stmt]
        [status-stmt]
        [description-stmt]
        [reference-stmt]
    "}") stmtsep

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

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

refine-stmt = refine-keyword sep refine-arg-str optsep
    "{" stmtsep
        ;; these stmts can appear in any order
        *if-feature-stmt
        *must-stmt
        [presence-stmt]
        [default-stmt]
        [config-stmt]
        [mandatory-stmt]
        [min-elements-stmt]
```

```

        [max-elements-stmt]
        [description-stmt]
        [reference-stmt]
    "}" stmtsep

refine-arg-str    = < a string that matches the rule >
                   < refine-arg >

refine-arg        = descendant-schema-nodeid

uses-augment-stmt = augment-keyword sep uses-augment-arg-str optsep
    "{" stmtsep
        ;; these stmts can appear in any order
        [when-stmt]
        *if-feature-stmt
        [status-stmt]
        [description-stmt]
        [reference-stmt]
        1*(data-def-stmt / case-stmt /
            action-stmt / notification-stmt)
    "}" stmtsep

uses-augment-arg-str = < a string that matches the rule >
                      < uses-augment-arg >

uses-augment-arg    = descendant-schema-nodeid

augment-stmt        = augment-keyword sep augment-arg-str optsep
    "{" stmtsep
        ;; these stmts can appear in any order
        [when-stmt]
        *if-feature-stmt
        [status-stmt]
        [description-stmt]
        [reference-stmt]
        1*(data-def-stmt / case-stmt /
            action-stmt / notification-stmt)
    "}" stmtsep

augment-arg-str    = < a string that matches the rule >
                   < augment-arg >

augment-arg        = absolute-schema-nodeid

when-stmt          = when-keyword sep string optsep
    ( ";" /
    "{" stmtsep
        ;; these stmts can appear in any order
```



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

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

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

input-stmt = input-keyword optsep
    "{" stmtsep
        ;; these stmts can appear in any order
        *must-stmt
        *(typedef-stmt / grouping-stmt)
        1*data-def-stmt
    "}" stmtsep

output-stmt = output-keyword optsep
    "{" stmtsep
        ;; these stmts can appear in any order
        *must-stmt
        *(typedef-stmt / grouping-stmt)
        1*data-def-stmt
    "}" stmtsep

notification-stmt = notification-keyword sep
    identifier-arg-str optsep
```

```

( ";" /
  "{" stmtsep
    ;; these stmts can appear in any order
    *if-feature-stmt
    *must-stmt
    [status-stmt]
    [description-stmt]
    [reference-stmt]
    *(typedef-stmt / grouping-stmt)
    *data-def-stmt
  "}") stmtsep

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

deviation-arg-str = < a string that matches the rule >
< deviation-arg >

deviation-arg = absolute-schema-nodeid

deviate-not-supported-stmt =
  deviate-keyword sep
  not-supported-keyword-str stmtend

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

deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep

```

```
( ";" /
  "{" stmtsep
    ;; these stmts can appear in any order
    [units-stmt]
    *must-stmt
    *unique-stmt
    [default-stmt]
  "}") stmtsep

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

not-supported-keyword-str = < a string that matches the rule >
                           < not-supported-keyword >

add-keyword-str           = < a string that matches the rule >
                           < add-keyword >

delete-keyword-str        = < a string that matches the rule >
                           < delete-keyword >

replace-keyword-str       = < a string that matches the rule >
                           < replace-keyword >

;; represents the usage of an extension statement
unknown-statement         = prefix ":" identifier [sep string] optsep
( ";" /
  "{" optsep
    *((yang-stmt / unknown-statement) optsep)
  "}") stmtsep

yang-stmt                 = action-stmt /
                           anydata-stmt /
                           anyxml-stmt /
                           argument-stmt /
                           augment-stmt /
                           base-stmt /
                           belongs-to-stmt /
```

bit-stmt /
case-stmt /
choice-stmt /
config-stmt /
contact-stmt /
container-stmt /
default-stmt /
description-stmt /
deviate-add-stmt /
deviate-delete-stmt /
deviate-not-supported-stmt /
deviate-replace-stmt /
deviation-stmt /
enum-stmt /
error-app-tag-stmt /
error-message-stmt /
extension-stmt /
feature-stmt /
fraction-digits-stmt /
grouping-stmt /
identity-stmt /
if-feature-stmt /
import-stmt /
include-stmt /
input-stmt /
key-stmt /
leaf-list-stmt /
leaf-stmt /
length-stmt /
list-stmt /
mandatory-stmt /
max-elements-stmt /
min-elements-stmt /
modifier-stmt /
module-stmt /
must-stmt /
namespace-stmt /
notification-stmt /
ordered-by-stmt /
organization-stmt /
output-stmt /
path-stmt /
pattern-stmt /
position-stmt /
prefix-stmt /
presence-stmt /
range-stmt /
reference-stmt /

```
refine-stmt /  
require-instance-stmt /  
revision-date-stmt /  
revision-stmt /  
rpc-stmt /  
status-stmt /  
submodule-stmt /  
typedef-stmt /  
type-stmt /  
unique-stmt /  
units-stmt /  
uses-augment-stmt /  
uses-stmt /  
value-stmt /  
when-stmt /  
yang-version-stmt /  
yin-element-stmt
```

;; Ranges

```
range-arg-str      = < a string that matches the rule >  
                   < range-arg >  
  
range-arg          = range-part *(optsep "|" optsep range-part)  
  
range-part         = range-boundary  
                   [optsep ".." optsep range-boundary]  
  
range-boundary     = min-keyword / max-keyword /  
                   integer-value / decimal-value
```

;; Lengths

```
length-arg-str     = < a string that matches the rule >  
                   < length-arg >  
  
length-arg         = length-part *(optsep "|" optsep length-part)  
  
length-part        = length-boundary  
                   [optsep ".." optsep length-boundary]  
  
length-boundary    = min-keyword / max-keyword /  
                   non-negative-integer-value
```

;; Date

```
date-arg-str       = < a string that matches the rule >  
                   < date-arg >
```

```
date-arg          = 4DIGIT "-" 2DIGIT "-" 2DIGIT

;; Schema Node Identifiers

schema-nodeid     = absolute-schema-nodeid /
                    descendant-schema-nodeid

absolute-schema-nodeid = 1*("/") node-identifier)

descendant-schema-nodeid =
    node-identifier
    [absolute-schema-nodeid]

node-identifier    = [prefix ":"] identifier

;; Instance Identifiers

instance-identifier = 1*("/") (node-identifier *predicate))

predicate          = "[" *WSP (predicate-expr / pos) *WSP "]"

predicate-expr     = (node-identifier / ".") *WSP "=" *WSP
                    ((DQUOTE string DQUOTE) /
                     (SQUOTE string SQUOTE))

pos                = non-negative-integer-value

;; leafref path

path-arg-str       = < a string that matches the rule >
                    < path-arg >

path-arg           = absolute-path / relative-path

absolute-path      = 1*("/") (node-identifier *path-predicate))

relative-path      = 1*("../" "/") descendant-path

descendant-path    = node-identifier
                    [*path-predicate absolute-path]

path-predicate     = "[" *WSP path-equality-expr *WSP "]"

path-equality-expr = node-identifier *WSP "=" *WSP path-key-expr

path-key-expr      = current-function-invocation *WSP "/" *WSP
```

```
rel-path-keyexpr

rel-path-keyexpr    = 1*("." *WSP "/" *WSP)
                    *(node-identifier *WSP "/" *WSP)
                    node-identifier

;;; Keywords, using RFC 7405 syntax for case-sensitive strings

;; statement keywords
action-keyword      = %s"action"
anydata-keyword     = %s"anydata"
anyxml-keyword      = %s"anyxml"
argument-keyword    = %s"argument"
augment-keyword     = %s"augment"
base-keyword        = %s"base"
belongs-to-keyword  = %s"belongs-to"
bit-keyword         = %s"bit"
case-keyword        = %s"case"
choice-keyword      = %s"choice"
config-keyword      = %s"config"
contact-keyword     = %s"contact"
container-keyword   = %s"container"
default-keyword     = %s"default"
description-keyword = %s"description"
enum-keyword        = %s"enum"
error-app-tag-keyword = %s"error-app-tag"
error-message-keyword = %s"error-message"
extension-keyword   = %s"extension"
deviation-keyword   = %s"deviation"
deviate-keyword     = %s"deviate"
feature-keyword     = %s"feature"
fraction-digits-keyword = %s"fraction-digits"
grouping-keyword    = %s"grouping"
identity-keyword    = %s"identity"
if-feature-keyword  = %s"if-feature"
import-keyword      = %s"import"
include-keyword     = %s"include"
input-keyword       = %s"input"
key-keyword         = %s"key"
leaf-keyword        = %s"leaf"
leaf-list-keyword   = %s"leaf-list"
length-keyword      = %s"length"
list-keyword        = %s"list"
mandatory-keyword   = %s"mandatory"
max-elements-keyword = %s"max-elements"
min-elements-keyword = %s"min-elements"
modifier-keyword    = %s"modifier"
module-keyword      = %s"module"
```

```
must-keyword           = %s"must"
namespace-keyword      = %s"namespace"
notification-keyword= %s"notification"
ordered-by-keyword    = %s"ordered-by"
organization-keyword= %s"organization"
output-keyword        = %s"output"
path-keyword          = %s"path"
pattern-keyword       = %s"pattern"
position-keyword      = %s"position"
prefix-keyword        = %s"prefix"
presence-keyword      = %s"presence"
range-keyword         = %s"range"
reference-keyword     = %s"reference"
refine-keyword        = %s"refine"
require-instance-keyword = %s"require-instance"
revision-keyword      = %s"revision"
revision-date-keyword = %s"revision-date"
rpc-keyword           = %s"rpc"
status-keyword        = %s"status"
submodule-keyword     = %s"submodule"
type-keyword          = %s"type"
typedef-keyword       = %s"typedef"
unique-keyword        = %s"unique"
units-keyword         = %s"units"
uses-keyword          = %s"uses"
value-keyword         = %s"value"
when-keyword          = %s"when"
yang-version-keyword= %s"yang-version"
yin-element-keyword  = %s"yin-element"
```

;; other keywords

```
add-keyword           = %s"add"
current-keyword       = %s"current"
delete-keyword        = %s"delete"
deprecated-keyword    = %s"deprecated"
false-keyword         = %s"false"
invert-match-keyword = %s"invert-match"
max-keyword           = %s"max"
min-keyword           = %s"min"
not-supported-keyword = %s"not-supported"
obsolete-keyword      = %s"obsolete"
replace-keyword       = %s"replace"
system-keyword        = %s"system"
true-keyword          = %s"true"
unbounded-keyword     = %s"unbounded"
user-keyword          = %s"user"
```



```

and-keyword      = %s"and"
or-keyword       = %s"or"
not-keyword      = %s"not"

```

```

current-function-invocation = current-keyword *WSP "(" *WSP ")"

```

```

;;; Basic Rules

```

```

prefix-arg-str    = < a string that matches the rule >
                  < prefix-arg >

```

```

prefix-arg        = prefix

```

```

prefix            = identifier

```

```

identifier-arg-str = < a string that matches the rule >
                  < identifier-arg >

```

```

identifier-arg     = identifier

```

```

;; An identifier MUST NOT start with (('X'|'x') ('M'|'m') ('L'|'l'))
identifier          = (ALPHA / "_")
                  *(ALPHA / DIGIT / "_" / "-" / ".")

```

```

identifier-ref-arg-str = < a string that matches the rule >
                      < identifier-ref-arg >

```

```

identifier-ref-arg  = identifier-ref

```

```

identifier-ref      = [prefix ":" ] identifier

```

```

string              = < an unquoted string as returned by >
                  < the scanner, that matches the rule >
                  < yang-string >

```

```

yang-string         = *yang-char

```

```

;; any Unicode or ISO/IEC 10646 character including tab, carriage
;; return, and line feed, but excluding the other C0 control
;; characters, the surrogate blocks, and the noncharacters.

```

```

yang-char = %x9 / %xA / %xD / %x20-D7FF /
            ; exclude surrogate blocks %xD800-DFFF
            %xE000-FDCF /      ; exclude noncharacters %xFDD0-FDEF
            %xFDF0-FFFD /      ; exclude noncharacters %xFFFE-FFFF
            %x10000-1FFFD /     ; exclude noncharacters %x1FFFE-1FFFF
            %x20000-2FFFD /     ; exclude noncharacters %x2FFFE-2FFFF
            %x30000-3FFFD /     ; exclude noncharacters %x3FFFE-3FFFF
            %x40000-4FFFD /     ; exclude noncharacters %x4FFFE-4FFFF

```

```

%x50000-5FFFD / ; exclude noncharacters %x5FFFE-5FFFF
%x60000-6FFFD / ; exclude noncharacters %x6FFFE-6FFFF
%x70000-7FFFD / ; exclude noncharacters %x7FFFE-7FFFF
%x80000-8FFFD / ; exclude noncharacters %x8FFFE-8FFFF
%x90000-9FFFD / ; exclude noncharacters %x9FFFE-9FFFF
%xA0000-AFFFD / ; exclude noncharacters %xAFFFE-AFFFF
%B0000-BFFFD / ; exclude noncharacters %xBFFFE-BFFFF
%C0000-CFFFD / ; exclude noncharacters %xCFFFE-CFFFF
%D0000-DFFFD / ; exclude noncharacters %xDFFFE-DFFFF
%E0000-EFFFD / ; exclude noncharacters %xEFFFE-EFFFF
%F0000-FFFFD / ; exclude noncharacters %xFFFFE-FFFFF
%x100000-10FFFD ; exclude noncharacters %x10FFFE-10FFFF

```

```

integer-value      = ("-" non-negative-integer-value) /
                    non-negative-integer-value

```

```

non-negative-integer-value = "0" / positive-integer-value

```

```

positive-integer-value = (non-zero-digit *DIGIT)

```

```

zero-integer-value  = 1*DIGIT

```

```

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

```

```

sep                = 1*(WSP / line-break)
                    ; unconditional separator

```

```

optsep             = *(WSP / line-break)

```

```

stmtsep            = *(WSP / line-break / unknown-statement)

```

```

line-break         = CRLF / LF

```

```

non-zero-digit     = %x31-39

```

```

decimal-value      = integer-value ( "." zero-integer-value )

```

```

SQUOTE            = %x27
                    ; ' (Single Quote)

```

```

;;; RFC 5234 core rules.

```

```

ALPHA              = %x41-5A / %x61-7A
                    ; A-Z / a-z

```

```

CR                 = %x0D
                    ; carriage return

```

CRLF = CR LF
; Internet standard new line

DIGIT = %x30-39
; 0-9

DQUOTE = %x22
; double quote

HEXDIG = DIGIT /
%x61 / %x62 / %x63 / %x64 / %x65 / %x66
; only lower-case a..f

HTAB = %x09
; horizontal tab

LF = %x0A
; linefeed

SP = %x20
; space

VCHAR = %x21-7E
; visible (printing) characters

WSP = SP / HTAB
; whitespace

<CODE ENDS>

15. NETCONF 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.

15.1. 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").

15.2. 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.

15.3. 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.

15.4. 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

15.5. 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" or "leafref" 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 or leafref leaf.
```

15.6. 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.
```

15.7. 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").

15.8. 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
```

16. IANA Considerations

This document registers one capability identifier URN from the "Network Configuration Protocol (NETCONF) Capability URNs" registry:

urn:ietf:params:netconf:capability:yang-library:1.0

17. 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.

18. 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)

19. 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, Per Hedeland, Leif Johansson, Ladislav Lhotka, Gerhard Muenz, Peyman Ovladi, Tom Petch, Randy Presuhn, David Reid, Jernej Tuljak, and Bert Wijnen.

20. ChangeLog

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

20.1. Version -08

- o Fixes after WGLC reviews.

20.2. Version -07

- o Fixes after WG review.
- o Included solution Y60-01.

20.3. Version -06

- o Included solution Y45-05.

20.4. Version -05

- o Included solution Y18-01.
- o Included solution Y25-02 (parts of it was included in -04).
- o Included solution Y34-05.
- o Included solution Y36-03.

20.5. Version -04

- o Incorporated changes to ABNF grammar after review and errata for RFC 6020.
- o Included solution Y16-03.

- o Included solution Y49-04.
- o Included solution Y58-01.
- o Included solution Y59-01.

20.6. Version -03

- o Incorporated changes from WG reviews.
- o Included solution Y05-01.
- o Included solution Y10-01.
- o Included solution Y13-01.
- o Included solution Y28-02.
- o Included solution Y55-01 (parts of it was included in -01).

20.7. Version -02

- o Included solution Y02-01.
- o Included solution Y04-02.
- o Included solution Y11-01.
- o Included solution Y41-01.
- o Included solution Y56-01.

20.8. Version -01

- o Included solution Y01-01.
- o Included solution Y03-01.
- o Included solution Y06-02.
- o Included solution Y07-01.
- o Included solution Y14-01.
- o Included solution Y20-01.
- o Included solution Y23-01.

- o Included solution Y29-01.
- o Included solution Y30-01.
- o Included solution Y31-01.
- o Included solution Y35-01.

20.9. Version -00

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

21. References

21.1. Normative References

- [I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library (work in progress), July 2015.
- [ISO.10646]
International Organization for Standardization,
"Information Technology - Universal Multiple-Octet Coded
Character Set (UCS)", ISO Standard 10646:2003, 2003.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
10646", STD 63, RFC 3629, November 2003.
- [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.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event
Notifications", RFC 5277, July 2008.

- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, December 2014.
- [XML-NAMES]
Hollander, D., Tobin, R., Thompson, H., Bray, T., and A. Layman, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009,
<<http://www.w3.org/TR/2009/REC-xml-names-20091208>>.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999,
<<http://www.w3.org/TR/1999/REC-xpath-19991116>>.
- [XSD-TYPES]
Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004,
<<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.

21.2. Informative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-04 (work in progress), June 2015.
- [I-D.vanderstok-core-comi]
Stok, P., Bierman, A., Schoenwaelder, J., and A. Sehgal, "CoAP Management Interface", draft-vanderstok-core-comi-07 (work in progress), July 2015.
- [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.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIV2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<http://www.rfc-editor.org/info/rfc6643>>.
- [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

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2016

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems
October 16, 2015

A YANG Data Model for Routing Management
draft-ietf-netmod-routing-cfg-20

Abstract

This document contains a specification of three YANG modules. Together they form the core routing data model which serves as a framework for configuring and managing a routing subsystem. It is expected that these modules will be augmented by additional YANG modules defining data models for routing protocols, route filters and other functions. The core routing data model provides common building blocks for such extensions - routing instances, routes, routing information bases (RIB), and routing protocols.

Status of This Memo

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 18, 2016.

Copyright Notice

Copyright (c) 2015 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. Terminology and Notation	3
2.1. Glossary of New Terms	4
2.2. Tree Diagrams	5
2.3. Prefixes in Data Node Names	5
3. Objectives	6
4. The Design of the Core Routing Data Model	6
4.1. System-Controlled and User-Controlled List Entries	8
5. Basic Building Blocks	8
5.1. Routing Instance	8
5.1.1. Parameters of IPv6 Router Interfaces	9
5.2. Route	10
5.3. Routing Information Base (RIB)	11
5.4. Routing Protocol	11
5.4.1. Routing Pseudo-Protocols	12
5.4.2. Defining New Routing Protocols	12
5.5. RPC Operations	13
6. Interactions with Other YANG Modules	13
6.1. Module "ietf-interfaces"	13
6.2. Module "ietf-ip"	13
7. Routing Management YANG Module	14
8. IPv4 Unicast Routing Management YANG Module	29
9. IPv6 Unicast Routing Management YANG Module	33
10. IANA Considerations	46
11. Security Considerations	47
12. Acknowledgments	48
13. References	48
13.1. Normative References	48
13.2. Informative References	49
Appendix A. The Complete Data Trees	49
A.1. Configuration Data	49
A.2. State Data	50
Appendix B. Minimum Implementation	51
Appendix C. Example: Adding a New Routing Protocol	52
Appendix D. Example: NETCONF <get> Reply	54
Appendix E. Change Log	61
E.1. Changes Between Versions -19 and -20	61
E.2. Changes Between Versions -18 and -19	61
E.3. Changes Between Versions -17 and -18	61
E.4. Changes Between Versions -16 and -17	62
E.5. Changes Between Versions -15 and -16	62

E.6.	Changes Between Versions -14 and -15	63
E.7.	Changes Between Versions -13 and -14	63
E.8.	Changes Between Versions -12 and -13	63
E.9.	Changes Between Versions -11 and -12	64
E.10.	Changes Between Versions -10 and -11	64
E.11.	Changes Between Versions -09 and -10	65
E.12.	Changes Between Versions -08 and -09	65
E.13.	Changes Between Versions -07 and -08	65
E.14.	Changes Between Versions -06 and -07	65
E.15.	Changes Between Versions -05 and -06	66
E.16.	Changes Between Versions -04 and -05	66
E.17.	Changes Between Versions -03 and -04	67
E.18.	Changes Between Versions -02 and -03	67
E.19.	Changes Between Versions -01 and -02	68
E.20.	Changes Between Versions -00 and -01	68
Authors' Addresses		69

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. It also augments the "ietf-interfaces" module [RFC7223] with IPv6 router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is intended as a basis for future data model development covering more sophisticated routing systems. While these three modules can be directly used for simple IP devices with static routing (see Appendix B), their main purpose is to provide essential building blocks for more complicated data models involving multiple routing protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- o client,
- o message,
- o protocol operation,
- o server.

The following terms are defined in [RFC6020]:

- o augment,
- o configuration data,
- o container,
- o container with presence,
- o data model,
- o data node,
- o feature,
- o leaf,
- o list,
- o mandatory node,
- o module,
- o schema tree,
- o state data,
- o RPC operation.

2.1. Glossary of New Terms

core routing data model: YANG data model comprising "ietf-routing", "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing" modules.

direct route: a route to a directly connected network.

routing information base (RIB): An object containing a list of routes together with other information. See Section 5.3 for details.

system-controlled entry: An entry of a list in state data ("config false") that is created by the system independently of what has been explicitly configured. See Section 4.1 for details.

user-controlled entry: An entry of a list in state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text.

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

2.3. Prefixes in Data Node Names

In this document, names of data nodes, RPC operations and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC7223]
ip	ietf-ip	[RFC7277]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

3. Objectives

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

- o The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).
- o A simple IP routing system, such as one that uses only 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 implementations involving multiple routing information bases (RIB) and multiple routing protocols, as well as controlled redistributions of routing information.
- o Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Figures 1 and 2 show abridged views of the configuration and state data hierarchies. See Appendix A for the complete data trees.

```

+--rw routing
  +--rw routing-instance* [name]
    +--rw name
    +--rw type?
    +--rw enabled?
    +--rw router-id?
    +--rw description?
    +--rw routing-protocols
      | +--rw routing-protocol* [type name]
      | | +--rw type
      | | +--rw name
      | | +--rw description?
      | | +--rw static-routes
      | | ...
    +--rw ribs
      +--rw rib* [name]
        +--rw name
        +--rw address-family?
        +--rw description?

```

Figure 1: Configuration data hierarchy.

```

+--ro routing-state
  +--ro routing-instance* [name]
    +--ro name
    +--ro type?
    +--ro router-id?
    +--ro interfaces
      | +--ro interface*
    +--ro routing-protocols
      | +--ro routing-protocol* [type name]
      | | +--ro type
      | | +--ro name
    +--ro ribs
      +--ro rib* [name]
        +--ro name
        +--ro address-family
        +--ro default-rib?
        +--ro routes
        ...

```

Figure 2: State data hierarchy.

As can be seen from Figures 1 and 2, the core routing data model introduces several generic components of a routing framework: routing instances, RIBs containing lists of routes, and routing protocols. Section 5 describes these components in more detail.

4.1. System-Controlled and User-Controlled List Entries

The core routing data model defines several lists in the schema tree, 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 a client.

In such a list, the server creates the required item as a so-called system-controlled entry in state data, i.e., inside the "routing-state" container.

Additional entries may be created in the configuration by a client, e.g., via the NETCONF protocol. These are so-called user-controlled entries. If the server accepts a configured user-controlled entry, then this entry also appears in the state data version of the list.

Corresponding entries in both versions of the list (in state data and configuration) have the same value of the list key.

A client may also provide supplemental configuration of system-controlled entries. To do so, the client creates a new entry in the configuration with the desired contents. In order to bind this entry to the corresponding entry in the state data list, the key of the configuration entry has to be set to the same value as the key of the state entry.

An example can be seen in Appendix D: the "/routing-state/routing-instance" list has a single system-controlled entry whose "name" key has the value "rtr0". This entry is configured by the "/routing/routing-instance" entry whose "name" key is also "rtr0".

Deleting a user-controlled entry from the configuration list results in the removal of the corresponding entry in the state data list. In contrast, if a system-controlled entry is deleted from the configuration list, only the extra configuration specified in that entry is removed but the corresponding state data entry remains in the list.

5. Basic Building Blocks

This section describes the essential components of the core routing data model.

5.1. Routing Instance

The core routing data model supports one or more routing instances appearing as entries of the "routing-instance" list. Each routing instance has separate configuration and state data under

"/rt:routing/rt:routing-instance" and "/rt:routing-state/rt:routing-instance", respectively.

No attempt has been made to define the semantics for every type of routing instance. The core routing data model defines identities for two ubiquitous routing instance types:

- o "default-routing-instance" - this routing instance type represents the default (or only) routing instance. All implementations MUST provide one and only one system-controlled routing instance of this type.
- o "vrf-routing-instance" - this routing instance type represents VRF (virtual routing and forwarding) routing instances that are used for virtual private networks (VPN) including BGP/MPLS VPN_[RFC4364].

It is expected that future YANG modules will define other types of routing instances. For every such type, an identity derived from "rt:routing-instance" SHALL be defined. This identity is then referred to by the value of the "type" leaf (a child node of "routing-instance" list).

By default, all network layer interfaces are assigned to the routing instance of the "default-routing-instance" type. This can be changed by configuring the "rt:routing-instance" leaf in the interface configuration.

5.1.1.1. Parameters of IPv6 Router Interfaces

YANG module "ietf-ipv6-unicast-routing" (Section 9) augments the configuration and state data of IPv6 interfaces 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.

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [RFC7277] (leaf "ip:forwarding").
2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-unicast-routing" module therefore assumes the former behavior with constant values.

5.2. Route

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

- o "destination-prefix": IP prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.
- o "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among

routes with the same destination prefix. A lower value means a more preferred route.

- o "next-hop": determines the action to be performed with a packet.

Routes are primarily state data that appear as entries of RIBs (Section 5.3) but they may also be found in configuration data, for example as manually configured static routes. In the latter case, configurable route attributes are generally a subset of route attributes described above.

5.3. Routing Information Base (RIB)

Every routing instance manages one or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing-state/routing-instance/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.

Each routing instance has, for every supported address family, one RIB marked as the so-called default RIB. Its role 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 routing instance and supported address family, and mark it as the default RIB.

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.

5.4. Routing Protocol

The core routing data model provides an open-ended framework for defining multiple routing protocol instances within a routing instance. Each routing protocol instance **MUST** be assigned a type, which is an identity derived from the "rt:routing-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.4.1).

Multiple routing protocol instances of the same type MAY be configured within the same routing instance.

5.4.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types - "direct" and "static". Both are in fact pseudo-protocols, which means they are confined to the local device and do not exchange any routing information with adjacent routers.

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. Direct routes MUST be installed in default RIBs of all supported address families.

A pseudo-protocol of the type "static" allows for specifying routes manually. It MAY be configured in zero or multiple instances, although a typical configuration will have exactly one instance per routing instance.

5.4.2. Defining New Routing Protocols

It is expected that future YANG modules will create data models for additional routing protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to fit it into the core routing framework in the following way:

- o A new identity MUST be defined for the routing protocol and its base identity MUST be set to "rt:routing-protocol", or to an identity derived from "rt:routing-protocol".
- o Additional route attributes MAY be defined, preferably in one place by means of defining a YANG grouping. The new attributes have to be inserted by augmenting the definitions of the nodes

/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route

and

/rt:fib-route/rt:output/rt:route,

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

- o Configuration parameters and/or state data for the new protocol can be defined by augmenting the "routing-protocol" data node under both "/routing" and "/routing-state".

By using a "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 an appropriately named container with presence. Such a container may contain mandatory data nodes that are otherwise forbidden at the top level of an augment.

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

5.5. RPC Operations

The "ietf-routing" module defines one RPC operation:

- o fib-route: query a routing instance for the active route in the Forwarding Information Base (FIB). It is the route that is currently used for sending datagrams to a destination host whose address is passed as an input parameter.

6. Interactions with Other YANG Modules

The semantics of the core routing data model also depends on several configuration parameters that are defined in other YANG modules.

6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [RFC7223]:

```
/if:interfaces/if:interface/if:enabled
```

If this switch is set to "false" for a network layer interface, then all routing and forwarding functions MUST be disabled on that interface.

6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [RFC7277]:

```
/if:interfaces/if:interface/ip:ipv4/ip:enabled
```


If this switch is set to "false" for a network layer interface, then all IPv4 routing and forwarding functions MUST be disabled on that interface.

```
/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 through 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 and forwarding functions MUST be disabled on that interface.

```
/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 through this interface MUST be disabled. However, the interface MAY participate in other IPv6 routing functions, such as routing protocols.

In addition, the "ietf-ip" module allows for configuring IPv4 and IPv6 addresses and network prefixes or masks on network layer interfaces. Configuration of these parameters on an enabled interface MUST result in an immediate creation of the corresponding direct route. The destination prefix of this route is set according to the configured IP address and network prefix/mask, and the interface is set as the outgoing interface for that route.

7. Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-routing@2015-10-16.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>

  WG Chair: Kent Watsen
             <mailto:kwatsen@juniper.net>

  Editor:     Ladislav Lhotka
             <mailto:lhotka@nic.cz>";

description
  "This YANG module defines essential components for the management
  of a routing subsystem.

  Copyright (c) 2015 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).

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
  'OPTIONAL' in the module text are to be interpreted as described
  in RFC 2119 (http://tools.ietf.org/html/rfc2119).

  This version of this YANG module is part of RFC XXXX
  (http://tools.ietf.org/html/rfcXXXX); see the RFC itself for
  full legal notices."
```

```
revision 2015-10-16 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Features */

feature multiple-ribs {
  description
    "This feature indicates that the server supports user-defined
    RIBs.

    Servers that do not advertise this feature SHOULD provide
    exactly one system-controlled RIB per routing-instance and
    supported address family and make them also the default RIBs.
    These RIBs then appear as entries of the list
    /routing-state/routing-instance/ribs/rib.";
}

feature router-id {
  description
    "This feature indicates that the server supports configuration
    of an explicit 32-bit router ID that is used by some routing
    protocols.

    Servers that do not advertise this feature set a router ID
    algorithmically, usually to one of configured IPv4 addresses.
    However, this algorithm is implementation-specific.";
}

/* Identities */

identity address-family {
  description
    "Base identity from which identities describing address
    families are derived.";
}

identity ipv4 {
  base address-family;
  description
    "This identity represents IPv4 address family.";
}

identity ipv6 {
  base address-family;
```

```
    description
      "This identity represents IPv6 address family.";
  }

  identity routing-instance {
    description
      "Base identity from which identities describing routing
       instance types are derived.";
  }

  identity default-routing-instance {
    base routing-instance;
    description
      "This identity represents either a default routing instance, or
       the only routing instance on systems that do not support
       multiple instances.";
  }

  identity vrf-routing-instance {
    base routing-instance;
    description
      "This identity represents a VRF routing instance. The type is
       distinct from the default-routing-instance. There may be
       multiple vrf-routing-interfaces.";
  }

  identity routing-protocol {
    description
      "Base identity from which routing protocol identities are
       derived.";
  }

  identity direct {
    base routing-protocol;
    description
      "Routing pseudo-protocol that provides routes to directly
       connected networks.";
  }

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

  /* Type Definitions */

  typedef routing-instance-ref {
```

```
    type leafref {
      path "/rt:routing/rt:routing-instance/rt:name";
    }
    description
      "This type is used for leafs that reference a routing instance
      configuration.";
  }

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

  typedef route-preference {
    type uint32;
    description
      "This type is used for route preferences.";
  }

  /* Groupings */

  grouping address-family {
    description
      "This grouping provides a leaf identifying an address
      family.";
    leaf address-family {
      type identityref {
        base address-family;
      }
      mandatory "true";
      description
        "Address family.";
    }
  }

  grouping router-id {
    description
      "This grouping provides router ID.";
    leaf router-id {
      type yang:dotted-quad;
      description
        "A 32-bit number in the form of a dotted quad that is used by
        some routing protocols identifying a router.";
      reference

```

```
        "RFC 2328: OSPF Version 2.";
    }
}

grouping special-next-hop {
    description
        "This grouping provides a leaf with an enumeration of special
        next-hops.";
    leaf special-next-hop {
        type enumeration {
            enum blackhole {
                description
                    "Silently discard the packet.";
            }
            enum unreachable {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the destination host is
                    unreachable.";
            }
            enum prohibit {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the communication is
                    administratively prohibited.";
            }
            enum receive {
                description
                    "The packet will be received by the local system.";
            }
        }
    }
    description
        "Special next-hop options.";
}

grouping next-hop-content {
    description
        "Generic parameters of next-hops in static routes.";
    choice next-hop-options {
        mandatory "true";
        description
            "Options for next-hops in static routes.

            Modules for address families MUST augment this choice with
            the 'next-hop-address' case, which is a leaf containing a
            gateway address of that address family."
    }
}
```

```
        It is expected that further cases will be added through
        augments from other modules, e.g., for Equal-Cost Multipath
        routing (ECMP).";
    leaf outgoing-interface {
        type if:interface-ref;
        description
            "Name of the outgoing interface.";
    }
    case special-next-hop {
        uses special-next-hop;
    }
}

grouping next-hop-state-content {
    description
        "Generic parameters of next-hops in state data.";
    choice next-hop-options {
        mandatory "true";
        description
            "Options for next-hops in state data.

            Modules for address families MUST augment this choice with
            the 'next-hop-address' case, which is a leaf containing a
            gateway address of that address family.

            It is expected that further cases will be added through
            augments from other modules, e.g., for ECMP or recursive
            next-hops.";
        leaf outgoing-interface {
            type if:interface-state-ref;
            description
                "Name of the outgoing interface.";
        }
        case special-next-hop {
            uses special-next-hop;
        }
    }
}

grouping route-metadata {
    description
        "Common route metadata.";
    leaf source-protocol {
        type identityref {
            base routing-protocol;
        }
        mandatory "true";
    }
}
```

```
    description
      "Type of the routing protocol from which the route
       originated.";
  }
  leaf active {
    type empty;
    description
      "Presence of this leaf indicates that the route is preferred
       among all routes in the same RIB that have the same
       destination prefix.";
  }
  leaf last-updated {
    type yang:date-and-time;
    description
      "Time stamp of the last modification of the route. If the
       route was never modified, it is the time when the route was
       inserted into the RIB.";
  }
}

/* State data */

augment "/if:interfaces-state/if:interface" {
  description
    "This augment adds a reference to the routing-instance to which
     the interface is assigned.";
  leaf routing-instance {
    type routing-instance-state-ref;
    description
      "The name of the routing instance to which the interface is
       assigned.";
  }
}

container routing-state {
  config "false";
  description
    "State data of the routing subsystem.";
  list routing-instance {
    key "name";
    min-elements "1";
    description
      "Each list entry is a container for state data of a routing
       instance.

       An implementation MUST provide one and only one
       system-controlled routing instance(s) of the type
       'rt:default-routing-instance', and MAY support other types.
```



```
    An implementation MAY restrict the number of routing
    instances of each supported type.";
  leaf name {
    type string;
    description
      "The name of the routing instance.

      For system-controlled instances the name SHOULD be
      persistent, i.e., it doesn't change after a reboot.";
  }
  leaf type {
    type identityref {
      base routing-instance;
    }
    description
      "The routing instance type.";
  }
  uses router-id {
    description
      "Global router ID.

      It may be either configured or assigned algorithmically by
      the implementation.";
  }
  container interfaces {
    description
      "Network layer interfaces belonging to the routing
      instance.";
    leaf-list interface {
      type if:interface-state-ref;
      must "../..name = /if:interfaces-state/"
        + "if:interface[if:name=current()]/"
        + "rt:routing-instance" {
        error-message
          "Routing instance is not assigned to the interface.";
        description
          "This reference must mirror a corresponding assignment
          of the ancestor routing-instance to the interface.";
      }
      description
        "Each entry is a reference to the name of a configured
        network layer interface.";
    }
  }
  container routing-protocols {
    description
      "Container for the list of routing protocol instances.";
    list routing-protocol {
```

```
key "type name";
description
  "State data of a routing protocol instance.

  An implementation MUST provide exactly one
  system-controlled instance of the type 'direct'. Other
  instances MAY be created by configuration.";
leaf type {
  type identityref {
    base routing-protocol;
  }
  description
    "Type of the routing protocol.";
}
leaf name {
  type string;
  description
    "The name of the routing protocol instance.

    For system-controlled instances this name is
    persistent, i.e., it SHOULD NOT change across
    reboots.";
}
}
container ribs {
  description
    "Container for RIBs.";
  list rib {
    key "name";
    min-elements "1";
    description
      "Each entry represents a RIB identified by the 'name'
      key. All routes in a RIB MUST belong to the same address
      family.

      For each routing instance, an implementation SHOULD
      provide one system-controlled default RIB for each
      supported address family.";
    leaf name {
      type string;
      description
        "The name of the RIB.";
    }
    uses address-family;
    leaf default-rib {
      if-feature multiple-ribs;
      type boolean;
```

```

    default "true";
    description
        "This flag has the value of 'true' if and only if the
        RIB is the default RIB for the given address family.

        A default RIB always receives direct routes. By
        default it also receives routes from all routing
        protocols.";
    }
    container routes {
        description
            "Current content of the RIB.";
        list route {
            description
                "A RIB route entry. This data node MUST be augmented
                with information specific for routes of each address
                family.";
            leaf route-preference {
                type route-preference;
                description
                    "This route attribute, also known as administrative
                    distance, allows for selecting the preferred route
                    among routes with the same destination prefix. A
                    smaller value means a more preferred route.";
            }
            container next-hop {
                description
                    "Route's next-hop attribute.";
                uses next-hop-state-content;
            }
            uses route-metadata;
        }
    }
}

/* Configuration Data */

augment "/if:interfaces/if:interface" {
    description
        "This augment adds a routing-instance reference to interface
        configuration.";
    leaf routing-instance {
        type routing-instance-ref;
        description
            "The name of the routing instance to which the interface is

```

to be assigned.

By default, all network layer interfaces belong to the routing-instance of the 'default-routing-instance' type.";

```
}
}

container routing {
  description
    "Configuration parameters for the routing subsystem.";
  list routing-instance {
    key "name";
    description
      "Configuration of a routing instance.";
    leaf name {
      type string;
      description
        "The name of the routing instance.

        For system-controlled entries, the value of this leaf must
        be the same as the name of the corresponding entry in
        state data.

        For user-controlled entries, an arbitrary name can be
        used.";
    }
    leaf type {
      type identityref {
        base routing-instance;
      }
      default "rt:default-routing-instance";
      description
        "The type of the routing instance.";
    }
    leaf enabled {
      type boolean;
      default "true";
      description
        "Enable/disable the routing instance.

        If this parameter is false, the parent routing instance is
        disabled and does not appear in state data, despite any
        other configuration that might be present.";
    }
    uses router-id {
      if-feature router-id;
      description
        "Configuration of the global router ID. Routing protocols
```

```
        that use router ID can use this parameter or override it
        with another value.";
    }
    leaf description {
        type string;
        description
            "Textual description of the routing instance.";
    }
    container routing-protocols {
        description
            "Configuration of routing protocol instances.";
        list routing-protocol {
            key "type name";
            description
                "Each entry contains configuration of a routing protocol
                instance.";
            leaf type {
                type identityref {
                    base routing-protocol;
                }
                description
                    "Type of the routing protocol - an identity derived
                    from the 'routing-protocol' base identity.";
            }
            leaf name {
                type string;
                description
                    "An arbitrary name of the routing protocol instance.";
            }
            leaf description {
                type string;
                description
                    "Textual description of the routing protocol
                    instance.";
            }
        }
        container static-routes {
            when "../type='rt:static'" {
                description
                    "This container is only valid for the 'static'
                    routing protocol.";
            }
            description
                "Configuration of the 'static' pseudo-protocol.

                Address-family-specific modules augment this node with
                their lists of routes.";
        }
    }
}
```

```

    }
    container ribs {
        description
            "Configuration of RIBs.";
        list rib {
            key "name";
            description
                "Each entry contains configuration for a RIB identified
                by the 'name' key.

                Entries having the same key as a system-controlled entry
                of the list /routing-state/routing-instance/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 {
                description
                    "Address family of the RIB.

                    It is mandatory for user-controlled RIBs. For
                    system-controlled RIBs it can be omitted, otherwise it
                    must match the address family of the corresponding
                    state entry.";
                refine "address-family" {
                    mandatory "false";
                }
            }
            leaf description {
                type string;
                description
                    "Textual description of the RIB.";
            }
        }
    }
}

```

```
/* RPC operations */

rpc fib-route {
  description
    "Return the active FIB route that a routing-instance uses for
    sending packets to a destination address.";
  input {
    leaf routing-instance-name {
      type routing-instance-state-ref;
      mandatory "true";
      description
        "Name of the routing instance whose forwarding information
        base is being queried.

        If the routing instance with name equal to the value of
        this parameter doesn't exist, then this operation SHALL
        fail with error-tag 'data-missing' and error-app-tag
        'routing-instance-not-found'.";
    }
    container destination-address {
      description
        "Network layer destination address.

        Address family specific modules MUST augment this
        container with a leaf named 'address'.";
      uses address-family;
    }
  }
  output {
    container route {
      description
        "The active FIB route for the specified destination.

        If the routing instance has no active FIB route for the
        destination address, no output is returned - the server
        SHALL send an <rpc-reply> containing a single element
        <ok>.

        Address family specific modules MUST augment this list
        with appropriate route contents.";
      uses address-family;
      container next-hop {
        description
          "Route's next-hop attribute.";
        uses next-hop-state-content;
      }
      uses route-metadata;
    }
  }
}
```

```
    }  
  }  
}
```

<CODE ENDS>

8. IPv4 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv4-unicast-routing@2015-10-16.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>  
  
    WG Chair:   Kent Watsen  
                <mailto:kwatsen@juniper.net>  
  
    Editor:     Ladislav Lhotka  
                <mailto:lhotka@nic.cz>";  
  
  description
```


"This YANG module augments the 'ietf-routing' module with basic configuration and state data for IPv4 unicast routing.

Copyright (c) 2015 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>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<http://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<http://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices."

```
revision 2015-10-16 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Identities */

identity ipv4-unicast {
  base rt:ipv4;
  description
    "This identity represents the IPv4 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:routing-instance/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:routing-instance/rt:ribs/rt:rib/"
  + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options" {
  when "../.../rt:address-family = 'v4ur:ipv4-unicast'" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "Augment 'next-hop-options' in IPv4 unicast routes.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

/* Configuration data */

augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
  + "rt:routing-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
    pseudo-protocol with data specific to IPv4 unicast.";
  container ipv4 {
    description
      "Configuration of a 'static' pseudo-protocol instance
      consists of a list of routes.";
    list route {
      key "destination-prefix";
      description
        "A list of static routes.";
      leaf destination-prefix {
        type inet:ipv4-prefix;
        mandatory "true";
        description
          "IPv4 destination prefix.";
      }
      leaf description {
        type string;
        description
          "Textual description of the route.";
      }
    }
    container next-hop {
```

```

        description
            "Configuration of next-hop.";
        uses rt:next-hop-content {
            augment "next-hop-options" {
                description
                    "Augment 'next-hop-options' in IPv4 static routes.";
                leaf next-hop-address {
                    type inet:ipv4-address;
                    description
                        "IPv4 address of the next-hop.";
                }
            }
        }
    }
}

/* RPC operations */

augment "/rt:fib-route/rt:input/rt:destination-address" {
    when "rt:address-family='v4ur:ipv4-unicast'" {
        description
            "This augment is valid only for IPv4 unicast.";
    }
    description
        "This leaf augments the 'rt:destination-address' parameter of
        the 'rt:fib-route' operation.";
    leaf address {
        type inet:ipv4-address;
        description
            "IPv4 destination address.";
    }
}

augment "/rt:fib-route/rt:output/rt:route" {
    when "rt:address-family='v4ur:ipv4-unicast'" {
        description
            "This augment is valid only for IPv4 unicast.";
    }
    description
        "This leaf augments the reply to the 'rt:fib-route'
        operation.";
    leaf destination-prefix {
        type inet:ipv4-prefix;
        description
            "IPv4 destination prefix.";
    }
}

```

```
    }

    augment "/rt:fib-route/rt:output/rt:route/rt:next-hop/"
      + "rt:next-hop-options" {
        when "../rt:address-family='v4ur:ipv4-unicast'" {
          description
            "This augment is valid only for IPv4 unicast.";
        }
        description
          "Augment 'next-hop-options' in the reply to the 'rt:fib-route'
          operation.";
        leaf next-hop-address {
          type inet:ipv4-address;
          description
            "IPv4 address of the next-hop.";
        }
      }
    }
  }
}
```

<CODE ENDS>

9. IPv6 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv6-unicast-routing@2015-10-16.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>

  WG Chair:   Kent Watsen
              <mailto:kwatsen@juniper.net>

  Editor:     Ladislav Lhotka
              <mailto:lhotka@nic.cz>";

description
  "This YANG module augments the 'ietf-routing' module with basic
  configuration and state data for IPv6 unicast routing.

  Copyright (c) 2015 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).

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
  'OPTIONAL' in the module text are to be interpreted as described
  in RFC 2119 (http://tools.ietf.org/html/rfc2119).

  This version of this YANG module is part of RFC XXXX
  (http://tools.ietf.org/html/rfcXXXX); see the RFC itself for
  full legal notices.";

revision 2015-10-16 {
  description
    "Initial revision.";
```

```
    reference
      "RFC XXXX: A YANG Data Model for Routing Management";
  }

/* Identities */

identity ipv6-unicast {
  base rt:ipv6;
  description
    "This identity represents the IPv6 unicast address family.";
}

/* State data */

augment "/if:interfaces-state/if:interface/ip:ipv6" {
  description
    "Augment interface state data with 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.

        An implementation SHOULD keep this value constant in
        consecutive advertisements except when it is
        explicitly changed in configuration.";
    }
    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.

        An implementation SHOULD keep this value constant in
        consecutive advertisements except when it is
        explicitly changed in configuration.";
    }
    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:routing-instance/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:routing-instance/rt:ribs/rt:rib/"
    + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options" {
    when "../../../rt:address-family = 'v6ur:ipv6-unicast'" {
        description
            "This augment is valid only for IPv6 unicast.";
    }
    description
        "Augment 'next-hop-options' in IPv6 unicast routes.";
    leaf next-hop-address {
        type inet:ipv6-address;
        description
            "IPv6 address of the next-hop.";
    }
}

/* Configuration data */

```

```
augment "/if:interfaces/if:interface/ip:ipv6" {
  description
    "Augment interface configuration with IPv6-specific parameters
    of router interfaces.";
  container ipv6-router-advertisements {
    description
      "Configuration of IPv6 Router Advertisements.";
    leaf send-advertisements {
      type boolean;
      default "false";
      description
        "A flag indicating whether or not the router sends periodic
        Router Advertisements and responds to Router
        Solicitations.";
      reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvSendAdvertisements.";
    }
    leaf max-rtr-adv-interval {
      type uint16 {
        range "4..1800";
      }
      units "seconds";
      default "600";
      description
        "The maximum time allowed between sending unsolicited
        multicast Router Advertisements from the interface.";
      reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        MaxRtrAdvInterval.";
    }
    leaf min-rtr-adv-interval {
      type uint16 {
        range "3..1350";
      }
      units "seconds";
      must ". <= 0.75 * ../max-rtr-adv-interval" {
        description
          "The value MUST NOT be greater than 75 % of
          'max-rtr-adv-interval'.";
      }
    }
    description
      "The minimum time allowed between sending unsolicited
      multicast Router Advertisements from the interface.

      The default value to be used operationally if this leaf is
      not configured is determined as follows:
```

```
- if max-rtr-adv-interval >= 9 seconds, the default value
  is 0.33 * max-rtr-adv-interval;

- otherwise it is 0.75 * max-rtr-adv-interval.";
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
  MinRtrAdvInterval.";
}
leaf managed-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Managed address
    configuration' flag field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvManagedFlag.";
}
leaf other-config-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Other configuration' flag
    field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvOtherConfigFlag.";
}
leaf link-mtu {
  type uint32;
  default "0";
  description
    "The value to be placed in MTU options sent by the router.
    A value of zero indicates that no MTU options are sent.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvLinkMTU.";
}
leaf reachable-time {
  type uint32 {
    range "0..3600000";
  }
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Reachable Time field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).";
```

```
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvReachableTime.";
}
leaf retrans-timer {
  type uint32;
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Retrans Timer field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
      AdvRetransTimer.";
}
leaf cur-hop-limit {
  type uint8;
  description
    "The value to be placed in the Cur Hop Limit field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).

    If this parameter is not configured, the device SHOULD use
    the value specified in IANA Assigned Numbers that was in
    effect at the time of implementation.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
      AdvCurHopLimit.

      IANA: IP Parameters,
      http://www.iana.org/assignments/ip-parameters";
}
leaf default-lifetime {
  type uint16 {
    range "0..9000";
  }
  units "seconds";
  description
    "The value to be placed in the Router Lifetime field of
    Router Advertisements sent from the interface, in seconds.
    It MUST be either zero or between max-rtr-adv-interval and
    9000 seconds. A value of zero indicates that the router is
    not to be used as a default router. These limits may be
    overridden by specific documents that describe how IPv6
    operates over different link layers.

    If this parameter is not configured, the device SHOULD use
```

```
        a value of 3 * max-rtr-adv-interval.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvDefaultLifeTime.";
}
container prefix-list {
    description
        "Configuration of prefixes to be placed in Prefix
        Information options in Router Advertisement messages sent
        from the interface.

        Prefixes that are advertised by default but do not have
        their entries in the child 'prefix' list are advertised
        with the default values of all parameters.

        The link-local prefix SHOULD NOT be included in the list
        of advertised prefixes.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvPrefixList.";
    list prefix {
        key "prefix-spec";
        description
            "Configuration of an advertised prefix entry.";
        leaf prefix-spec {
            type inet:ipv6-prefix;
            description
                "IPv6 address prefix.";
        }
        choice control-adv-prefixes {
            default "advertise";
            description
                "The prefix either may be explicitly removed from the
                set of advertised prefixes, or parameters with which
                it is advertised may be specified (default case).";
            leaf no-advertise {
                type empty;
                description
                    "The prefix will not be advertised.

                    This can be used for removing the prefix from the
                    default set of advertised prefixes.";
            }
            case advertise {
                leaf valid-lifetime {
                    type uint32;
                    units "seconds";
                    default "2592000";
                }
            }
        }
    }
}
```

```
description
  "The value to be placed in the Valid Lifetime in
  the Prefix Information option. The designated
  value of all 1's (0xffffffff) represents
  infinity.";
reference
  "RFC 4861: Neighbor Discovery for IP version 6
  (IPv6) - AdvValidLifetime.";
}
leaf on-link-flag {
  type boolean;
  default "true";
  description
    "The value to be placed in the on-link flag
    ('L-bit') field in the Prefix Information
    option.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6
    (IPv6) - AdvOnLinkFlag.";
}
leaf preferred-lifetime {
  type uint32;
  units "seconds";
  must ". <= ../valid-lifetime" {
    description
      "This value MUST NOT be greater than
      valid-lifetime.";
  }
  default "604800";
  description
    "The value to be placed in the Preferred Lifetime
    in the Prefix Information option. The designated
    value of all 1's (0xffffffff) represents
    infinity.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6
    (IPv6) - AdvPreferredLifetime.";
}
leaf autonomous-flag {
  type boolean;
  default "true";
  description
    "The value to be placed in the Autonomous Flag
    field in the Prefix Information option.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6
    (IPv6) - AdvAutonomousFlag.";
}
```

```

    }
  }
}

augment "/rt:routing/rt:routing-instance/rt:routing-protocols/"
+ "rt:routing-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
    pseudo-protocol with data specific to IPv6 unicast.";
  container ipv6 {
    description
      "Configuration of a 'static' pseudo-protocol instance
      consists of a list of routes.";
    list route {
      key "destination-prefix";
      description
        "A list of static routes.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        mandatory "true";
        description
          "IPv6 destination prefix.";
      }
      leaf description {
        type string;
        description
          "Textual description of the route.";
      }
      container next-hop {
        description
          "Configuration of next-hop.";
        uses rt:next-hop-content {
          augment "next-hop-options" {
            description
              "Augment 'next-hop-options' in IPv6 static routes.";
            leaf next-hop-address {
              type inet:ipv6-address;
              description
                "IPv6 address of the next-hop.";
            }
          }
        }
      }
    }
  }
}

```

```
}

/* RPC operations */

augment "/rt:fib-route/rt:input/rt:destination-address" {
  when "rt:address-family='v6ur:ipv6-unicast'" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "This leaf augments the 'rt:destination-address' parameter of
    the 'rt:fib-route' operation.";
  leaf address {
    type inet:ipv6-address;
    description
      "IPv6 destination address.";
  }
}

augment "/rt:fib-route/rt:output/rt:route" {
  when "rt:address-family='v6ur:ipv6-unicast'" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "This leaf augments the reply to the 'rt:fib-route'
    operation.";
  leaf destination-prefix {
    type inet:ipv6-prefix;
    description
      "IPv6 destination prefix.";
  }
}

augment "/rt:fib-route/rt:output/rt:route/rt:next-hop/"
  + "rt:next-hop-options" {
  when "../rt:address-family='v6ur:ipv6-unicast'" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "Augment 'next-hop-options' in the reply to the 'rt:fib-route'
    operation.";
  leaf next-hop-address {
    type inet:ipv6-address;
    description
      "IPv6 address of the next-hop.";
  }
}
```



```
}  
}
```

<CODE ENDS>

10. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URIs in the IETF XML registry [RFC3688]:

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

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing

Registrant Contact: The IESG.

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

This document registers the following YANG modules in the YANG Module Names registry [RFC6020]:

```
-----
name:          ietf-routing
namespace:     urn:ietf:params:xml:ns:yang:ietf-routing
prefix:        rt
reference:     RFC XXXX
-----
```

```
-----
name:          ietf-ipv4-unicast-routing
namespace:     urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
prefix:        v4ur
reference:     RFC XXXX
-----
```

```
-----
name:          ietf-ipv6-unicast-routing
namespace:     urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
prefix:        v6ur
reference:     RFC XXXX
-----
```

11. Security Considerations

Configuration and state data conforming to the core routing data model (defined in this document) are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

A number of data nodes defined in the YANG modules belonging to the configuration part of the core routing data model are writable/creatable/deletable (i.e., "config true" in YANG terms, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes, such as "edit-config", can have negative effects on the network if the protocol operations are not properly protected.

The vulnerable "config true" parameters and subtrees are the following:

/if:interfaces/if:interface/rt:routing-instance: This leaf assigns a network layer interface to a routing instance.

/routing/routing-instance/routing-protocols/routing-protocol: This list specifies the routing protocols configured on a device.

/routing/routing-instance/ribs/rib: This list specifies the RIBs configured for the device.

Unauthorised access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

12. Acknowledgments

The authors wish to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Jeff Haas, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Stephane Litkowski, Thomas Morin, Tom Petch, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler, Yi Yang, Derek Man-Kit Yeung and Jeffrey Zhang for their helpful comments and suggestions.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<http://www.rfc-editor.org/info/rfc4861>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.

- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.

13.2. Informative References

- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<http://www.rfc-editor.org/info/rfc4364>>.
- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <<http://www.rfc-editor.org/info/rfc6087>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Appendix A. The Complete Data Trees

This appendix presents the complete configuration and state data trees of the core routing data model. See Section 2.2 for an explanation of the symbols used. Data type of every leaf node is shown near the right end of the corresponding line.

A.1. Configuration Data

```

+--rw routing
  +--rw routing-instance* [name]
    +--rw name string
    +--rw type? identityref
    +--rw enabled? boolean
    +--rw router-id? yang:dotted-quad
    +--rw description? string
    +--rw routing-protocols
      +--rw routing-protocol* [type name]
        +--rw type identityref
        +--rw name string
        +--rw description? string
        +--rw static-routes
          +--rw v6ur:ipv6
            +--rw v6ur:route* [destination-prefix]
              +--rw v6ur:destination-prefix inet:ipv6-prefix
              +--rw v6ur:description? string
              +--rw v6ur:next-hop
                +--rw (next-hop-options)
                  +--:(outgoing-interface)
                  | +--rw v6ur:outgoing-interface?
                  +--:(special-next-hop)
                  | +--rw v6ur:special-next-hop?
                  +--:(next-hop-address)
                  +--rw v6ur:next-hop-address?
          +--rw v4ur:ipv4
            +--rw v4ur:route* [destination-prefix]
              +--rw v4ur:destination-prefix inet:ipv4-prefix
              +--rw v4ur:description? string
              +--rw v4ur:next-hop
                +--rw (next-hop-options)
                  +--:(outgoing-interface)
                  | +--rw v4ur:outgoing-interface?
                  +--:(special-next-hop)
                  | +--rw v4ur:special-next-hop?
                  +--:(next-hop-address)
                  +--rw v4ur:next-hop-address?
        +--rw ribs
          +--rw rib* [name]
            +--rw name string
            +--rw address-family? identityref
            +--rw description? string

```

A.2. State Data

```

+--ro routing-state
  +--ro routing-instance* [name]
    +--ro name                string
    +--ro type?               identityref
    +--ro router-id?          yang:dotted-quad
    +--ro interfaces
      | +--ro interface*      if:interface-state-ref
    +--ro routing-protocols
      | +--ro routing-protocol* [type name]
      |   +--ro type          identityref
      |   +--ro name          string
    +--ro ribs
      +--ro rib* [name]
        +--ro name            string
        +--ro address-family  identityref
        +--ro default-rib?    boolean {multiple-ribs}?
        +--ro routes
          +--ro route*
            +--ro route-preference?      route-preference
            +--ro next-hop
              | +--ro (next-hop-options)
              |   +--:(outgoing-interface)
              |   | +--ro outgoing-interface?
              |   | +--:(special-next-hop)
              |   |   +--ro special-next-hop?      enumeration
              |   |   +--:(next-hop-address)
              |   |   +--ro v6ur:next-hop-address?
              |   |   +--:(next-hop-address)
              |   |   +--ro v4ur:next-hop-address?
              |   +--ro source-protocol            identityref
              +--ro active?                        empty
              +--ro last-updated?                  yang:date-and-time
              +--ro v6ur:destination-prefix?      inet:ipv6-prefix
              +--ro v4ur:destination-prefix?      inet:ipv4-prefix

```

Appendix B. Minimum Implementation

Some parts and options of the core routing model, such as user-defined RIBs, 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 provides a single system-controlled routing instance of the type "default-routing-instance", and will not allow clients to create any user-controlled instances.

Typically, the feature "multiple-ribs" will not be supported. This means that a single system-controlled RIB is available for each supported address family - IPv4, IPv6 or both. These RIBs must be the default RIBs. No user-controlled RIBs are allowed.

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configuring instance(s) of the "static" pseudo-protocol.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" routing protocol instances.

Appendix C. Example: Adding a New Routing Protocol

This appendix demonstrates how the core routing data model can be extended to support a new routing protocol. The YANG module "example-rip" shown below is intended as an illustration rather than a real definition of a data model for the RIP routing protocol. For the sake of brevity, this module does not obey all the guidelines specified in [RFC6087]. See also Section 5.4.2.

```
module example-rip {  
    namespace "http://example.com/rip";  
  
    prefix "rip";  
  
    import ietf-interfaces {  
        prefix "if";  
    }  
  
    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:routing-instance/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:fib-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 {
    presence "RIP configuration";
    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 if:interface-ref;
  }
  leaf enabled {
    type boolean;
    default "true";
  }
  leaf metric {
    type rip-metric;
    default "1";
  }
}
}
leaf update-interval {
  type uint8 {
    range "10..60";
  }
  units "seconds";
  default "30";
  description
    "Time interval between periodic updates.";
}
}
}
```

Appendix D. Example: NETCONF <get> Reply

This section contains a sample reply to the NETCONF <get> message, which could be sent by a server supporting (i.e., advertising them in the NETCONF <hello> message) the following YANG modules:

- o ietf-interfaces [RFC7223],
- o ietf-ip [RFC7277],
- o ietf-routing (Section 7),
- o ietf-ipv4-unicast-routing (Section 8),
- o ietf-ipv6-unicast-routing (Section 9).

We assume a simple network set-up as shown in Figure 3: 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.

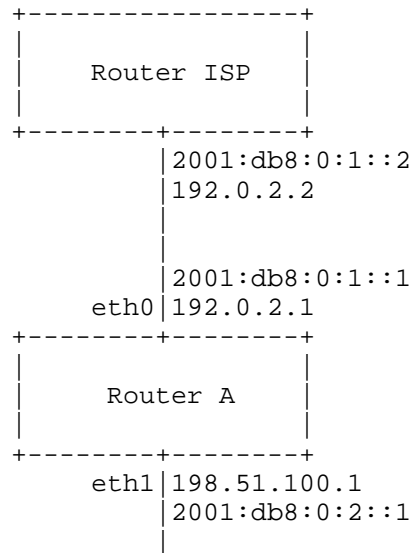


Figure 3: Example network configuration

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>
  
```

```
<rt:routing-instance>rtr0</rt:routing-instance>
<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>
  <rt:routing-instance>rtr0</rt:routing-instance>
  <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>
<rt:routing-instance>rtr0</rt:routing-instance>
<if:statistics>
  <if:discontinuity-time>
    2015-10-24T17:11:27+02:00
  </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>
  <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>
</ip:ipv6>
</if:interface>
<if:interface>
  <if:name>eth1</if:name>
  <if:type>ianaift:ethernetCsmacd</if:type>
  <if:phys-address>00:0C:42:E5:B1:EA</if:phys-address>
  <if:oper-status>up</if:oper-status>
  <rt:routing-instance>rtr0</rt:routing-instance>
  <if:statistics>
    <if:discontinuity-time>
      2015-10-24T17:11:29+02:00
    </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>
  <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>
</ip:ipv6>
</if:interface>
</if:interfaces-state>
<rt:routing>
  <rt:routing-instance>
    <rt:name>rtr0</rt:name>
    <rt:description>Router A</rt:description>
    <rt:router-id>192.0.2.1</rt:router-id>
    <rt:routing-protocols>
      <rt:routing-protocol>
        <rt:type>rt:static</rt:type>
        <rt:name>st0</rt:name>
        <rt:description>
          Static routing is used for the internal network.
        </rt:description>
      <rt:static-routes>
        <v4ur:ipv4>
          <v4ur:route>
            <v4ur:destination-prefix>
              0.0.0.0/0
            </v4ur:destination-prefix>
            <v4ur:next-hop>
              <v4ur:next-hop-address>192.0.2.2</v4ur:next-hop-address>
            </v4ur:next-hop>
          </v4ur:route>
        </v4ur:ipv4>
        <v6ur:ipv6>
          <v6ur:route>
```

```
<v6ur:destination-prefix>::/0</v6ur:destination-prefix>
<v6ur:next-hop>
  <v6ur:next-hop-address>
    2001:db8:0:1::2
  </v6ur:next-hop-address>
</v6ur:next-hop>
</v6ur:route>
</v6ur:ipv6>
</rt:static-routes>
</rt:routing-protocol>
</rt:routing-protocols>
</rt:routing-instance>
</rt:routing>
<rt:routing-state>
  <rt:routing-instance>
    <rt:name>rtr0</rt:name>
    <rt:interfaces>
      <rt:interface>eth0</rt:interface>
      <rt:interface>eth1</rt:interface>
    </rt:interfaces>
    <rt:routing-protocols>
      <rt:routing-protocol>
        <rt:type>rt:static</rt:type>
        <rt:name>st0</rt:name>
      </rt:routing-protocol>
    </rt:routing-protocols>
    <rt:ribs>
      <rt:rib>
        <rt:name>ipv4-master</rt:name>
        <rt:address-family>v4ur:ipv4-unicast</rt:address-family>
        <rt:default-rib>true</rt:default-rib>
        <rt:routes>
          <rt:route>
            <v4ur:destination-prefix>
              192.0.2.1/24
            </v4ur:destination-prefix>
            <rt:next-hop>
              <rt:outgoing-interface>eth0</rt:outgoing-interface>
            </rt:next-hop>
            <rt:route-preference>0</rt:route-preference>
            <rt:source-protocol>rt:direct</rt:source-protocol>
            <rt:last-updated>2015-10-24T17:11:27+02:00</rt:last-updated>
          </rt:route>
          <rt:route>
            <v4ur:destination-prefix>
              198.51.100.0/24
            </v4ur:destination-prefix>
            <rt:next-hop>
```

```
<rt:outgoing-interface>eth1</rt:outgoing-interface>
</rt:next-hop>
<rt:source-protocol>rt:direct</rt:source-protocol>
<rt:route-preference>0</rt:route-preference>
<rt:last-updated>2015-10-24T17:11:27+02:00</rt:last-updated>
</rt:route>
<rt:route>
  <v4ur:destination-prefix>0.0.0.0/0</v4ur:destination-prefix>
  <rt:source-protocol>rt:static</rt:source-protocol>
  <rt:route-preference>5</rt:route-preference>
  <rt:next-hop>
    <v4ur:next-hop-address>192.0.2.2</v4ur:next-hop-address>
  </rt:next-hop>
  <rt:last-updated>2015-10-24T18:02:45+02:00</rt:last-updated>
</rt:route>
</rt:routes>
</rt:rib>
<rt:rib>
  <rt:name>ipv6-master</rt:name>
  <rt:address-family>v6ur:ipv6-unicast</rt:address-family>
  <rt:default-rib>true</rt:default-rib>
  <rt:routes>
    <rt:route>
      <v6ur:destination-prefix>
        2001:db8:0:1::/64
      </v6ur:destination-prefix>
      <rt:next-hop>
        <rt:outgoing-interface>eth0</rt:outgoing-interface>
      </rt:next-hop>
      <rt:source-protocol>rt:direct</rt:source-protocol>
      <rt:route-preference>0</rt:route-preference>
      <rt:last-updated>2015-10-24T17:11:27+02:00</rt:last-updated>
    </rt:route>
    <rt:route>
      <v6ur:destination-prefix>
        2001:db8:0:2::/64
      </v6ur:destination-prefix>
      <rt:next-hop>
        <rt:outgoing-interface>eth1</rt:outgoing-interface>
      </rt:next-hop>
      <rt:source-protocol>rt:direct</rt:source-protocol>
      <rt:route-preference>0</rt:route-preference>
      <rt:last-updated>2015-10-24T17:11:27+02:00</rt:last-updated>
    </rt:route>
    <rt:route>
      <v6ur:destination-prefix>::/0</v6ur:destination-prefix>
      <rt:next-hop>
        <v6ur:next-hop-address>
```

```
        2001:db8:0:1::2
      </v6ur:next-hop-address>
    </rt:next-hop>
    <rt:source-protocol>rt:static</rt:source-protocol>
    <rt:route-preference>5</rt:route-preference>
    <rt:last-updated>2015-10-24T18:02:45+02:00</rt:last-updated>
  </rt:route>
</rt:routes>
</rt:rib>
</rt:ribs>
</rt:routing-instance>
</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 -19 and -20

- o Assignment of L3 interfaces to routing instances is now part of interface configuration.
- o Next-hop options in configuration were aligned with state data.
- o It is recommended to enclose protocol-specific configuration in a presence container.

E.2. Changes Between Versions -18 and -19

- o The leaf "route-preference" was removed from the "routing-protocol" container in both "routing" and "routing-state".
- o The "vrf-routing-instance" identity was added in support of a common routing-instance type in addition to the "default-routing-instance".
- o Removed "enabled" switch from "routing-protocol".

E.3. Changes Between Versions -17 and -18

- o The container "ribs" was moved under "routing-instance" (in both "routing" and "routing-state").
- o Typedefs "rib-ref" and "rib-state-ref" were removed.
- o Removed "recipient-ribs" (both state and configuration).

- o Removed "connected-ribs" from "routing-protocol" (both state and configuration).
- o Configuration and state data for IPv6 RA were moved under "if:interface" and "if:interface-state".
- o Assignment of interfaces to routing instances now use leaf-list rather than list (both config and state). The opposite reference from "if:interface" to "rt:routing-instance" was changed to a single leaf (an interface cannot belong to multiple routing instances).
- o Specification of a default RIB is now a simple flag under "rib" (both config and state).
- o Default RIBs are marked by a flag in state data.

E.4. Changes Between Versions -16 and -17

- o Added Acee as a co-author.
- o Removed all traces of route filters.
- o Removed numeric IDs of list entries in state data.
- o Removed all next-hop cases except "simple-next-hop" and "special-next-hop".
- o Removed feature "multipath-routes".
- o Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing from state data of an interface entry to the routing instance(s) to which the interface is assigned.

E.5. Changes Between Versions -15 and -16

- o Added 'type' as the second key component of 'routing-protocol', both in configuration and state data.
- o The restriction of no more than one connected RIB per address family was removed.
- o Removed the 'id' key of routes in RIBs. This list has no keys anymore.
- o Remove the 'id' key from static routes and make 'destination-prefix' the only key.

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

E.6. Changes Between Versions -14 and -15

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

E.7. 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.8. Changes Between Versions -12 and -13

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

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

E.9. 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.10. 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.11. Changes Between Versions -09 and -10

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

E.12. 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.13. Changes Between Versions -07 and -08

- o Changed reference from RFC6021 to RFC6021bis.

E.14. 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.15. 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.16. 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.17. Changes Between Versions -03 and -04

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

E.18. 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.19. 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.20. Changes Between Versions -00 and -01

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

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

Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Lindem
Cisco Systems

Email: acee@cisco.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 9, 2016

L. Lhotka
CZ.NIC
October 07, 2015

JSON Encoding of Data Modeled with YANG
draft-ietf-netmod-yang-json-06

Abstract

This document defines encoding rules for representing configuration, state data, RPC operation or action input and output parameters, and notifications defined using YANG as JavaScript Object Notation (JSON) text.

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 9, 2016.

Copyright Notice

Copyright (c) 2015 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	3
3. Properties of the JSON Encoding	4
4. Names and Namespaces	4
5. Encoding of YANG Data Node Instances	6
5.1. The "leaf" Data Node	7
5.2. The "container" Data Node	7
5.3. The "leaf-list" Data Node	7
5.4. The "list" Data Node	8
5.5. The "anydata" Data Node	9
5.6. The "anyxml" Data Node	10
6. Representing YANG Data Types in JSON Values	10
6.1. Numeric Types	10
6.2. The "string" Type	11
6.3. The "boolean" Type	11
6.4. The "enumeration" Type	11
6.5. The "bits" Type	11
6.6. The "binary" Type	11
6.7. The "leafref" Type	12
6.8. The "identityref" Type	12
6.9. The "empty" Type	12
6.10. The "union" Type	13
6.11. The "instance-identifier" Type	14
7. I-JSON Compliance	14
8. Security Considerations	15
9. Acknowledgments	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Appendix A. A Complete Example	16
Appendix B. Change Log	18
B.1. Changes Between Revisions -05 and -06	18
B.2. Changes Between Revisions -04 and -05	18
B.3. Changes Between Revisions -03 and -04	19
B.4. Changes Between Revisions -02 and -03	19
B.5. Changes Between Revisions -01 and -02	19
B.6. Changes Between Revisions -00 and -01	19
Author's Address	20

1. Introduction

The NETCONF protocol [RFC6241] uses XML [W3C.REC-xml-20081126] for encoding data in its Content Layer. Other management protocols might want to use other encodings while still benefiting from using YANG [I-D.ietf-netmod-rfc6020bis] as the data modeling language.

For example, the RESTCONF protocol [I-D.ietf-netconf-restconf] supports two encodings: XML (media type "application/yang.data+xml") and JSON (media type "application/yang.data+json").

The specification of YANG 1.1 data modelling language [I-D.ietf-netmod-rfc6020bis] defines only XML encoding for data instances, i.e., contents of configuration datastores, state data, RPC operation or action input and output parameters, and event notifications. The aim of this document is to define rules for encoding the same data as JavaScript Object Notation (JSON) text [RFC7159].

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 [I-D.ietf-netmod-rfc6020bis]:

- o action,
- o anydata,
- o anyxml,
- o augment,
- o container,
- o data node,
- o data tree,
- o identity,
- o instance identifier,
- o leaf,
- o leaf-list,
- o list,
- o module,
- o RPC operation,

- o submodule.

3. Properties of the JSON Encoding

This document defines JSON encoding for YANG data trees and their subtrees. It is always assumed that the top-level structure in JSON-encoded data is an object.

Instances of YANG data nodes (leafs, containers, leaf-lists, lists, anydata and anyxml nodes) are encoded as members of a JSON object, i.e., name/value pairs. Section 4 defines how the name part is formed, and the following sections deal with the value part.

Unlike XML element content, JSON values carry partial type information (number, string, boolean). The JSON encoding is defined so that this information is never in conflict with the data type of the corresponding YANG leaf or leaf-list.

With the exception of anyxml and schema-less anydata nodes, it is possible to map a JSON-encoded data tree to XML encoding as defined in [I-D.ietf-netmod-rfc6020bis], and vice versa. However, such conversions require the YANG data model to be available.

In order to achieve maximum interoperability while allowing implementations to use a variety of existing JSON parsers, the JSON encoding rules follow, as much as possible, the constraints of the I-JSON restricted profile [RFC7493]. Section 7 discusses I-JSON conformance in more detail.

4. Names and Namespaces

An object member name MUST be in one of the following forms:

- o simple - identical to the identifier of the corresponding YANG data node;
- o namespace-qualified - the data node identifier is prefixed with the name of the module in which the data node is defined, separated from the data node identifier by the colon character (":").

The name of a module determines the namespace of all data node names defined in that module. If a data node is defined in a submodule, then the namespace-qualified member name uses the name of the main module to which the submodule belongs.

ABNF syntax [RFC5234] of a member name is shown in Figure 1, where the production for "identifier" is defined in sec. 13 of [I-D.ietf-netmod-rfc6020bis].

```
member-name = [identifier ":" ] identifier
```

Figure 1: ABNF production for a JSON member name.

A namespace-qualified member name MUST be used for all members of a top-level JSON object, and then also whenever the namespaces of the data node and its parent node are different. In all other cases, the simple form of the member name MUST be used.

For example, consider the following YANG module:

```
module foomod {  
  
    namespace "http://example.com/foomod";  
  
    prefix "foo";  
  
    container top {  
        leaf foo {  
            type uint8;  
        }  
    }  
}
```

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

```
{  
  "foomod:top": {  
    "foo": 54  
  }  
}
```

Note that the member of the top-level object uses the namespace-qualified name but the "foo" leaf doesn't because it is defined in the same module as its parent container "top".

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

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

A valid JSON-encoded configuration containing both leafs may then look like this:

```
{  
  "foomod:top": {  
    "foo": 54,  
    "barmod:bar": true  
  }  
}
```

The name of the "bar" leaf is prefixed with the namespace identifier because its parent is defined in a different module.

Explicit namespace identifiers are sometimes needed when encoding values of the "identityref" and "instances-identifier" types. The same form of namespace-qualified name as defined above is then used. See Sections 6.8 and 6.11 for details.

5. Encoding of YANG Data Node Instances

Every data node instance is encoded as a name/value pair where the name is formed from the data node identifier using the rules of Section 4. The value depends on the category of the data node as explained in the following subsections.

Character encoding MUST be UTF-8.

5.1. The "leaf" Data Node

A leaf instance is encoded as a name/value pair where the value can be a string, number, literal "true" or "false", or the special array "[null]", depending on the type of the leaf (see Section 6 for the type encoding rules).

Example: For the leaf node definition

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

the following is a valid JSON-encoded instance:

```
"foo": 123
```

5.2. The "container" Data Node

A container instance is encoded as a name/object pair. The container's child data nodes are encoded as members of the object.

Example: For the container definition

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

the following is a valid JSON-encoded instance:

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

5.3. The "leaf-list" Data Node

A leaf-list is encoded as a name/array pair, and the array elements are values of some scalar type, which can be a string, number, literal "true" or "false", or the special array "[null]", depending on the type of the leaf-list (see Section 6 for the type encoding rules).

The ordering of array elements follows the same rules as the ordering of XML elements representing leaf-list entries in the XML encoding. Specifically, the "ordered-by" properties (sec. 7.7.7 in [I-D.ietf-netmod-rfc6020bis]) MUST be observed.

Example: For the leaf-list definition

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

the following is a valid JSON-encoded instance:

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

5.4. The "list" Data Node

A list instance is encoded as a name/array pair, and the array elements are JSON objects.

The ordering of array elements follows the same rules as the ordering of XML elements representing list entries in the XML encoding. Specifically, the "ordered-by" properties (sec. 7.7.7 in [I-D.ietf-netmod-rfc6020bis]) MUST be observed.

Unlike the XML encoding, where list keys are required to precede any other siblings within a list entry, and appear in the order specified by the data model, the order of members in a JSON-encoded list entry is arbitrary because JSON objects are fundamentally unordered collections of members.

Example: For the list definition

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

the following is a valid JSON-encoded instance:


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

5.5. The "anydata" Data Node

Anydata data node serves as a container for an arbitrary set of nodes that otherwise appear as normal YANG-modeled data. A data model for anydata content may or may not be known at run time. In the latter case, converting JSON-encoded instances to the XML encoding defined in [I-D.ietf-netmod-rfc6020bis] may be impossible.

An anydata instance is encoded in the same way as a container, i.e., as a value/object pair. The requirement that anydata content can be modeled by YANG implies the following rules for the JSON text inside the object:

- o It is valid I-JSON [RFC7493].
- o All object member names satisfy the ABNF production in Figure 1.
- o Any JSON array contains either only unique scalar values (as a leaf-list, see Section 5.3), or only objects (as a list, see Section 5.4).
- o The "null" value is only allowed in the single-element array "[null]" corresponding to the encoding of the "empty" type, see Section 6.9.

Example: for the anydata definition

```
anydata data;
```

the following is a valid JSON-encoded instance:

```
"data": {  
  "ietf-notification:notification": {  
    "eventTime": "2014-07-29T13:43:01Z",  
    "example-event:event": {  
      "event-class": "fault",  
      "reporting-entity": {  
        "card": "Ethernet0"  
      },  
      "severity": "major"  
    },  
  }  
}
```

5.6. The "anyxml" Data Node

An anyxml instance is encoded as a JSON name/value pair which MUST satisfy I-JSON constraints. Otherwise it is unrestricted, i.e., the value can be an object, array, number, string or one of the literals "true", "false" and "null".

There is no universal procedure for mapping JSON-encoded anyxml instances to XML, and vice versa.

Example: For the anyxml definition

```
anyxml bar;
```

the following is a valid JSON-encoded instance:

```
"bar": [true, null, true]
```

6. Representing YANG Data Types in JSON Values

The type of the JSON value in an instance of the leaf or leaf-list data node depends on the type of that data node as specified in the following subsections.

6.1. Numeric Types

A value of the types "int8", "int16", "int32", "uint8", "uint16" and "uint32" is represented as a JSON number.

A value of the "int64", "uint64" or "decimal64" type is represented as a JSON string whose content is the lexical representation of the corresponding YANG type as specified in sections 9.2.1 and 9.3.1 of [I-D.ietf-netmod-rfc6020bis].

For example, if the type of the leaf "foo" in Section 5.1 was "uint64" instead of "uint8", the instance would have to be encoded as

```
"foo": "123"
```

The special handling of 64-bit numbers follows from the I-JSON recommendation to encode numbers exceeding the IEEE 754-2008 double precision range as strings, see sec. 2.2 in [RFC7493].

6.2. The "string" Type

A "string" value is represented as a JSON string, subject to JSON string encoding rules.

6.3. The "boolean" Type

A "boolean" value is represented as the corresponding JSON literal name "true" or "false".

6.4. The "enumeration" Type

An "enumeration" value is represented as a JSON string - one of the names assigned by "enum" statements in YANG.

The representation is identical to the lexical representation of the "enumeration" type in XML, see sec. 9.6 in [I-D.ietf-netmod-rfc6020bis].

6.5. The "bits" Type

A "bits" value is represented as a JSON string - a space-separated sequence of names of bits that are set. The permitted bit names are assigned by "bit" statements in YANG.

The representation is identical to the lexical representation of the "bits" type, see sec. 9.7 in [I-D.ietf-netmod-rfc6020bis].

6.6. The "binary" Type

A "binary" value is represented as a JSON string - base64-encoding of arbitrary binary data.

The representation is identical to the lexical representation of the "binary" type in XML, see sec. 9.8 in [I-D.ietf-netmod-rfc6020bis].

6.7. The "leafref" Type

A "leafref" value is represented using the same rules as the type of the leaf to which the leafref value refers.

6.8. The "identityref" Type

An "identityref" value is represented as a string - the name of an identity. If the identity is defined in another module than the leaf node containing the identityref value, the namespace-qualified form (Section 4) MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted.

For example, consider the following schematic module:

```
module exmod {  
  ...  
  import ietf-interfaces {  
    prefix if;  
  }  
  import iana-if-type {  
    prefix ianaift;  
  }  
  ...  
  leaf type {  
    type identityref {  
      base "if:interface-type";  
    }  
  }  
}
```

A valid instance of the "type" leaf is then encoded as follows:

```
"type": "iana-if-type:ethernetCsmacd"
```

The namespace identifier "iana-if-type" must be present in this case because the "ethernetCsmacd" identity is not defined in the same module as the "type" leaf.

6.9. The "empty" Type

An "empty" value is represented as "[null]", i.e., an array with the "null" literal being its only element. For the purposes of this document, "[null]" is considered an atomic scalar value.

This encoding of the "empty" type was chosen instead of using simply "null" in order to facilitate the use of empty leafs in common

programming languages where the "null" value of a member is treated as if the member is not present.

Example: For the leaf definition

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

a valid instance is

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

6.10. The "union" Type

A value of the "union" type is encoded as the value of any of the member types.

When validating a value of the "union" type, the type information conveyed by the JSON encoding MUST also be taken into account. JSON syntax thus provides additional means for resolving union member type that are not available in XML encoding.

For example, consider the following YANG definition:

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

In RESTCONF [I-D.ietf-netconf-restconf], it is possible to set the value of "bar" in the following way when using the "application/yang.data+xml" media type:

```
<bar>13.5</bar>
```

because the value may be interpreted as a string, i.e., the second member type of the union. When using the "application/yang.data+json" media type, however, this is an error:

```
"bar": 13.5
```

In this case, the JSON encoding indicates the value is supposed to be a number rather than a string, and it is not a valid "uint16" value.

Conversely, the value of

```
"bar": "1"
```

is to be interpreted as a string.

6.11. The "instance-identifier" Type

An "instance-identifier" value is encoded as a string that is analogical to the lexical representation in XML encoding, see sec. 9.13.3 in [I-D.ietf-netmod-rfc6020bis]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 4, namely:

- o The leftmost (top-level) data node name is always in the namespace-qualified form.
- o Any subsequent data node name is in the namespace-qualified form if the node is defined in another module than its parent node, and the simple form is used otherwise. This rule also holds for node names appearing in predicates.

For example,

```
/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip
```

is a valid instance-identifier value because the data nodes "interfaces", "interface" and "name" are defined in the module "ietf-interfaces", whereas "ipv4" and "ip" are defined in "ietf-ip".

7. I-JSON Compliance

I-JSON [RFC7493] is a restricted profile of JSON that guarantees maximum interoperability for protocols that use JSON in their messages, no matter what JSON encoders/decoders are used in protocol implementations. The encoding defined in this document therefore observes the I-JSON requirements and recommendations as closely as possible.

In particular, the following properties are guaranteed:

- o Character encoding is UTF-8.
- o Member names within the same JSON object are always unique.
- o The order of JSON object members is never relied upon.
- o Numbers of any type supported by YANG can be exchanged reliably. See Section 6.1 for details.

The JSON encoding defined in this document deviates from I-JSON only in the representation of the "binary" type. In order to remain compatible with XML encoding, the base64 encoding scheme is used (Section 6.6), whilst I-JSON recommends base64url instead.

8. Security Considerations

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, it doesn't contribute any new security issues beyond those discussed in sec. 16 of [I-D.ietf-netmod-rfc6020bis].

JSON processing is rather different from XML, and JSON parsers may thus suffer from other types of vulnerabilities than their XML counterparts. To minimize these new security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

9. Acknowledgments

The author wishes to thank Andy Bierman, Martin Bjorklund, Dean Bogdanovic, Balazs Lengyel, Juergen Schoenwaelder and Phil Shafer for their helpful comments and suggestions.

10. References

10.1. Normative References

- [I-D.ietf-netmod-rfc6020bis] Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-07 (work in progress), September 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.

- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<http://www.rfc-editor.org/info/rfc7493>>.

10.2. Informative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.

Appendix A. A Complete Example

The JSON document shown below represents the same data as the reply to the NETCONF <get> request appearing in Appendix D of [RFC7223]. The data model is a combination of two YANG modules: "ietf-interfaces" and "ex-vlan" (the latter is an example module from Appendix C of [RFC7223]). The "if-mib" feature defined in the "ietf-interfaces" module is considered to be active.

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ex-vlan:vlan-tagging": true
      }
    ]
  }
}
```



```
    },
    {
      "name": "eth1.10",
      "type": "iana-if-type:l2vlan",
      "enabled": true,
      "ex-vlan:base-interface": "eth1",
      "ex-vlan:vlan-id": 10
    },
    {
      "name": "lo1",
      "type": "iana-if-type:softwareLoopback",
      "enabled": true
    }
  ]
},
"ietf-interfaces:interfaces-state": {
  "interface": [
    {
      "name": "eth0",
      "type": "iana-if-type:ethernetCsmacd",
      "admin-status": "down",
      "oper-status": "down",
      "if-index": 2,
      "phys-address": "00:01:02:03:04:05",
      "statistics": {
        "discontinuity-time": "2013-04-01T03:00:00+00:00"
      }
    },
    {
      "name": "eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "admin-status": "up",
      "oper-status": "up",
      "if-index": 7,
      "phys-address": "00:01:02:03:04:06",
      "higher-layer-if": [
        "eth1.10"
      ],
      "statistics": {
        "discontinuity-time": "2013-04-01T03:00:00+00:00"
      }
    },
    {
      "name": "eth1.10",
      "type": "iana-if-type:l2vlan",
      "admin-status": "up",
      "oper-status": "up",
      "if-index": 9,
```

```
    "lower-layer-if": [
      "eth1"
    ],
    "statistics": {
      "discontinuity-time": "2013-04-01T03:00:00+00:00"
    }
  },
  {
    "name": "eth2",
    "type": "iana-if-type:ethernetCsmacd",
    "admin-status": "down",
    "oper-status": "down",
    "if-index": 8,
    "phys-address": "00:01:02:03:04:07",
    "statistics": {
      "discontinuity-time": "2013-04-01T03:00:00+00:00"
    }
  },
  {
    "name": "lol",
    "type": "iana-if-type:softwareLoopback",
    "admin-status": "up",
    "oper-status": "up",
    "if-index": 1,
    "statistics": {
      "discontinuity-time": "2013-04-01T03:00:00+00:00"
    }
  }
]
}
```

Appendix B. Change Log

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

B.1. Changes Between Revisions -05 and -06

- o More text and a new example about resolving union-type values.

B.2. Changes Between Revisions -04 and -05

- o Removed section "Validation of JSON-encoded Instance Data" and other text about XML-JSON mapping.
- o Added section "Properties of the JSON Encoding".

B.3. Changes Between Revisions -03 and -04

- o I-D.ietf-netmod-rfc6020bis is used as a normative reference instead of RFC 6020.
- o Removed noncharacters as an I-JSON issue because it doesn't exist in YANG 1.1.
- o Section about anydata encoding was added.
- o Require I-JSON for anyxml encoding.
- o Use ABNF for defining qualified name.

B.4. Changes Between Revisions -02 and -03

- o Namespace encoding is defined without using RFC 2119 keywords.
- o Specification for anyxml nodes was extended and clarified.
- o Text about ordering of list entries was corrected.

B.5. Changes Between Revisions -01 and -02

- o Encoding of namespaces in instance-identifiers was changed.
- o Text specifying the order of array elements in leaf-list and list instances was added.

B.6. Changes Between Revisions -00 and -01

- o Metadata encoding was moved to a separate I-D, draft-lhotka-netmod-yang-metadata.
- o JSON encoding is now defined directly rather than via XML-JSON mapping.
- o The rules for namespace encoding has changed. This affect both node instance names and instance-identifiers.
- o I-JSON-related changes. The most significant is the string encoding of 64-bit numbers.
- o When validating union type, the partial type info present in JSON encoding is taken into account.
- o Added section about I-JSON compliance.

- o Updated the example in appendix.
- o Wrote Security Considerations.
- o Removed IANA Considerations as there are none.

Author's Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 20, 2016

L. Lhotka
CZ.NIC
September 17, 2015

Defining and Using Metadata with YANG
draft-ietf-netmod-yang-metadata-02

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 20, 2016.

Copyright Notice

Copyright (c) 2015 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	4
2.1. Keywords	4
2.2. Terms Defined in Other Documents	4
2.3. Namespaces and Prefixes	6
2.4. Definitions of New Terms	6
3. Defining Annotations in YANG	6
3.1. Example Definition	7
4. Using Annotations	8
5. The Encoding of Annotations	9
5.1. XML Encoding	9
5.2. JSON Encoding	10
5.2.1. Metadata Object and Annotations	10
5.2.2. Adding Annotations to Anydata, Container and List Entries	11
5.2.3. Adding Annotations to Anyxml and Leaf Instances	11
5.2.4. Adding Annotations to Leaf-list Entries	12
6. Representing Annotations in DSDL Schemas	13
7. Metadata YANG Module	14
8. IANA Considerations	16
9. Security Considerations	17
10. Acknowledgments	17
11. References	17
11.1. Normative References	17
11.2. Informative References	18
Appendix A. Change Log	19
A.1. Changes Between Revisions -01 and -02	19
A.2. Changes Between Revisions -00 and -01	19
A.3. Changes Between draft-lhotka-netmod-yang-metadata-01 and draft-ietf-netmod-yang-metadata-00	19
A.4. Changes Between draft-lhotka-netmod-yang-metadata-00 and -01	19
Author's Address	20

1. Introduction

There is a need to be able to annotate instances of YANG [I-D.ietf-netmod-rfc6020bis] data nodes with metadata. Typical use cases are:

- o Complementing regular data model information with instance-specific metadata, comments etc.
- o Providing information about data rendering in user interfaces.

- o Deactivating a subtree in a configuration datastore while keeping the data in place.
- o Network management protocols often use metadata annotations for various purposes in both operation requests and responses. For example, the <edit-config> operation in the NETCONF protocol (see section 7.2 of [RFC6241]) uses annotations in the form of XML attributes for identifying the location in a configuration datastore and the type of the operation.

However, metadata annotations could potentially lead to interoperability problems if they are used in an ad hoc fashion by different parties and/or without proper documentation. A sound metadata framework for YANG should therefore satisfy these requirements:

1. The set of annotations must be extensible in a decentralised manner so as to allow for defining new annotations without running into the risk of collisions with annotations defined and used by others.
2. Syntax and semantics of annotations must be documented and the documentation must be easily accessible.
3. Clients of network management protocols such as NETCONF [RFC6241] or RESTCONF [I-D.ietf-netconf-restconf] must be able to discover all annotations supported by a given server and identify each of them correctly.
4. Annotations sent by a server should not break clients that don't support them.

This document proposes a systematic way for defining metadata annotations. For this purpose, YANG extension statement "annotation" is defined in the module "ietf-yang-metadata" (Section 7). Other YANG modules importing this module can use the "annotation" statement for defining one or more annotations.

The benefits of defining the metadata annotations in a YANG module are the following:

- o Each annotation is bound to a YANG module name, namespace URI and prefix. This makes its encoding in instance documents (both XML and JSON) straightforward and consistent with the encoding of YANG data node instances.
- o Annotations are indirectly registered through IANA in the "YANG Module Names" registry [RFC6020].

- o Annotations are included in the data model. YANG compilers and tools supporting a certain annotation can thus take them into account and modify their behavior accordingly.
- o Semantics of an annotation are defined in the "description" and "reference" statements.
- o An annotation can be declared as conditional by using the "if-feature" statement.
- o Values of annotations are not limited to strings; any YANG built-in or derived type may be used for them.

In the XML encoding, XML attributes are a natural instrument for attaching annotations to data node instances. This document deliberately adopts some restrictions in order to remain compatible with the XML encoding of YANG data node instances and limitations of XML attributes. Specifically,

- o annotations are scalar values and cannot be further structured;
- o annotations cannot be attached to a whole list or leaf-list instance, only to individual list or leaf-list entries.

2. Terminology

2.1. Keywords

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].

2.2. Terms Defined in Other Documents

The following terms are defined in [RFC6241]:

- o capability,
- o client,
- o datastore,
- o message,
- o protocol operation,
- o server.

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action,
- o anydata,
- o anyxml,
- o data type,
- o container,
- o data model,
- o data node,
- o data tree,
- o extension,
- o leaf,
- o leaf-list,
- o list,
- o module,
- o RPC input and output.

The following terms are defined in [W3C.REC-xml-infoset-20040204]:

- o attribute,
- o document,
- o element.

The following terms are defined in [W3C.REC-xml-names11-20060816]:

- o local name,
- o namespace name,
- o prefix,
- o qualified name.

The following terms are defined in [RFC7159]:

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

2.3. Namespaces and Prefixes

In the following text, XML element names and YANG extension statements are always written with explicit namespace prefixes that are assumed to be bound to URI references as shown in Table 1.

Prefix	URI Reference
elm	http://example.org/example-last-modified
md	urn:ietf:params:xml:ns:yang:ietf-yang-metadata
rng	http://relaxng.org/ns/structure/1.0

Table 1: Used namespace prefixes and corresponding URI references

2.4. Definitions of New Terms

- o annotation: a single item of metadata that is attached to YANG data node instances.
- o metadata: additional information that complements a data tree.
- o metadata object: an object in JSON encoding that contains all annotations attached to a given data node instance.

3. Defining Annotations in YANG

Metadata annotations are defined by YANG extension statement "md:annotation". This YANG language extension is defined in the module "ietf-yang-metadata" (Section 7).

Substatements of "md:annotation" are shown in Table 2. They are all core YANG statements, and the numbers in the second column refer to the corresponding section in [I-D.ietf-netmod-rfc6020bis] where each statement is described.

substatement	RFC 6020bis section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1

Table 2: Substatements of "md:annotation".

Using the "type" statement, a type is specified for the annotation value according to the same rules as for YANG "leaf" type.

An annotation can be made conditional by using one or more "if-feature" statements; the annotation is then supported only by servers that advertise the corresponding feature.

The semantics and usage rules for an annotation SHOULD be fully specified in "description", "reference" and "units" statements.

An annotation MUST NOT change the data tree semantics defined by YANG. For example, it is illegal to define and use an annotation that allows for overriding uniqueness of leaf-list entries.

The "status" statement can be used exactly as for YANG data nodes.

A YANG module containing one or more "md:annotation" extension statements SHOULD NOT be used for defining data nodes or groupings. Also, derived types, identities and features SHOULD NOT be defined in such a module unless they are used by the definitions of annotations in that module.

3.1. Example Definition

The following module defines the "last-modified" annotation:

```
module example-last-modified {  
  namespace "http://example.org/example-last-modified";  
  prefix "elm";  
  import ietf-yang-types {  
    prefix "yang";  
  }  
  import ietf-yang-metadata {  
    prefix "md";  
  }  
  md:annotation last-modified {  
    type yang:date-and-time;  
    description  
      "This annotation contains date and time when the  
        annotated instance was last modified (or created).";  
  }  
}
```

4. Using Annotations

By advertising a YANG module in which a metadata annotation is defined using the "md:annotation" statement, a server indicates that it is prepared to handle that annotation anywhere in any data tree that is a part of the server's data model (configuration, state data, RPC operation/action input or output). That is, an annotation advertised by the server may be attached to any instance of a data node defined in any YANG module that is also advertised by the server.

Depending on its semantics, an annotation may have an effect only in certain data trees and/or on instances of specific data nodes types. If such an annotation appears elsewhere, it is syntactically valid but the annotation is ignored.

A client **MUST NOT** use an annotation in protocol operations if the server didn't advertise it.

Annotations modify the schema of datastores and/or management protocol messages, and may also change their semantics. Therefore, due care has to be exercised when introducing annotations in network management systems in order to avoid interoperability problems and software failures. The following aspects should be taken into account:

- o A client may not be able to parse or validate protocol messages containing annotations.
- o A client may not understand semantics of an annotation set by the server or other clients.

Generally, it is safe for a server to use annotations in the following cases, provided that the client is able to parse them and discard those that it doesn't understand or support:

- o An annotation is an integral part of a built-in or negotiated protocol capability.
- o An annotation contains optional information that is not critical for protocol operation.
- o The client explicitly asks the server, e.g., via a parameter of a protocol operation request, for including an annotation in the response.

5. The Encoding of Annotations

XML attributes are a natural choice for encoding metadata in XML instance documents. For JSON [RFC7159], there is no generally established method for encoding metadata. This document thus introduces a special encoding method that is consistent with the JSON encoding of YANG data node instances as defined in [I-D.ietf-netmod-yang-json].

5.1. XML Encoding

Metadata annotations are added to XML-encoded instances of YANG data nodes as XML attributes according to these rules:

- o The local name of the attribute SHALL be the same as the name of the annotation specified in the argument of the corresponding "md:annotation" statement.
- o The namespace of the attribute SHALL be identified by the URI that appears as the argument of the "namespace" statement in the YANG module where the annotation is defined. It is RECOMMENDED that the prefix specified by the "prefix" statement in the same module is used in the qualified name of the attribute.
- o The attribute value SHALL be encoded in the same way as the value of a YANG leaf instance having the same type, see [I-D.ietf-netmod-rfc6020bis], sec. 9.

For example, the "last-modified" annotation defined in Section 3.1 may be encoded as follows:

```
<foo xmlns:elm="http://example.org/example-last-modified"
  elm:last-modified="2015-09-16T10:27:35+02:00">
  ...
</foo>
```

5.2. JSON Encoding

The JSON metadata encoding defined in this section has the following properties:

1. The encoding of YANG data node instances as defined in [I-D.ietf-netmod-yang-json] does not change.
2. Namespaces of metadata annotations are encoded in the same way as namespaces of YANG data node instances, see [I-D.ietf-netmod-yang-json].

5.2.1. Metadata Object and Annotations

All metadata annotations assigned to a YANG data node instance are encoded as members (name/value pairs) of a single JSON object, henceforth denoted as the metadata object. The placement and name of this object depends on the type of the data node as specified in the following subsections.

The name of a metadata annotation (as a member of the metadata object) has the following ABNF syntax [RFC5234], where the production for "identifier" is defined in sec. 13 of [I-D.ietf-netmod-rfc6020bis]

annotation-name = identifier ":" identifier

where the left identifier is the name of the YANG module in which the annotation is defined, and the identifier on the right is the name of the annotation specified in the argument of the corresponding "md:annotation" statement.

Note that unlike member names of YANG data node instances in JSON encoding (see sec. 4 in [I-D.ietf-netmod-yang-json]), for annotations the explicit namespace identifier (module name) must always be present.

The value of a metadata annotation SHALL be encoded in exactly the same way as the value of a YANG leaf node having the same type as the annotation, see [I-D.ietf-netmod-yang-json], sec. 6.

5.2.2. Adding Annotations to Anydata, Container and List Entries

For a data node instance that is encoded as a JSON object (i.e., a container, list entry, or anydata node), the metadata object is added as a new member of that object with the name "@".

Examples:

- o "cask" is a container or anydata node:

```
"cask": {
  "@": {
    "example-last-modified:last-modified":
      "2015-09-16T10:27:35+02:00"
  },
  ...
}
```

- o "seq" is a list whose key is "name", annotation "last-modified" is added only to the first entry:

```
"seq": [
  {
    "@": {
      "example-last-modified:last-modified":
        "2015-09-16T10:27:35+02:00"
    },
    "name": "one",
    ...
  },
  {
    "name": "two",
    ...
  }
]
```

5.2.3. Adding Annotations to Anyxml and Leaf Instances

For an anyxml or leaf instance, the metadata object is added as a sibling name/value pair whose name is the symbol "@" concatenated with the name of the leaf or anyxml member that is being annotated. The namespace part (module name) is included if and only if it is in the name of the annotated member.

Examples:

- o "flag" is a leaf node of the "boolean" type defined in module "foo", and we assume the namespace name has to be expressed in its JSON encoding:

```
"foo:flag": true,  
"@foo:flag": {  
  "example-last-modified:last-modified":  
    "2015-09-16T10:27:35+02:00"  
}
```

- o "stuff" is an anyxml node:

```
"stuff": [1, null, "three"],  
"@stuff": {  
  "example-last-modified:last-modified":  
    "2015-09-16T10:27:35+02:00"  
}
```

5.2.4. Adding Annotations to Leaf-list Entries

For a leaf-list entry, which is represented as a JSON array with values of a primitive type, annotations may be assigned to one or more entries by adding a name/array pair as a sibling of the leaf-list entry, where the name is symbol "@" concatenated with the name of the leaf-list that is being annotated, and the value is a JSON array whose i-th element is the metadata object with annotations assigned to the i-th entry of the leaf-list entry, or null if the i-th entry has no annotations.

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

For example, in the following leaf-list instance with four entries, the "last-modified" annotation is added to the second and third entry in the following way:

```
"bibliomod:folio": [6, 3, 7, 8],  
"@bibliomod:folio": [  
  null,  
  { "example-last-modified:last-modified":  
    "2015-06-18T17:01:14+02:00"  
  },  
  { "example-last-modified:last-modified":  
    "2015-09-16T10:27:35+02:00"  
  },  
]  
]
```


6. Representing Annotations in DSDL Schemas

[RFC6110] defines the standard mapping of YANG data models to Document Schema Definition Languages (DSDL) [ISO.19757-1]. This section specifies the mapping for the extension statement "md:annotation" (Section 7), which enables validation of XML instance documents containing metadata annotations.

The first step of the DSDL mapping procedure, i.e., the transformation of the YANG data model to the hybrid schema (see sec. 6 in [RFC6110]), is modified as follows:

1. If the data model contains at least one "md:annotation" statement, then a RELAX NG named pattern definition **MUST** be added as a child of the root <rng:grammar> element in the hybrid schema. It is **RECOMMENDED** to use the name "__yang_metadata__" for this named pattern.
2. A reference to the named pattern described in item 1 **MUST** be included as a child of every <rng:element> pattern that corresponds to an anydata, container, leaf, leaf-list or list data node.
3. Every metadata annotation definition in the form

```
md:annotation ARGUMENT {  
    ...  
}
```

is mapped to the following RELAX NG pattern:

```
<rng:optional>  
  <rng:attribute name="PREFIX:ARGUMENT">  
    ...  
  </rng:attribute>  
</rng:optional>
```

where PREFIX is the prefix bound to the namespace URI of the YANG module that contains the "md:annotation" statement. The above pattern **SHALL** be inserted as a child of the named pattern described in item 1.

4. Substatements of "md:annotation" **SHALL** be mapped to children of the "rng:attribute" pattern exactly as described in sec. 10 of [RFC6110].

For example, the named pattern (item 1), when constructed only for the "last-modified" annotation, will have the following definition:

```
<rng:define name="__yang_metadata__">
  <rng:optional>
    <rng:attribute name="elm:last-modified">
      <rng:ref name="ietf-yang-types__date-and-time"/>
    </rng:attribute>
  </rng:optional>
</rng:define>
```

Every "rng:element" pattern that corresponds to an anydata, container, leaf, list or leaf-list data node will then contain a reference to the above named pattern, for example

```
<rng:element name="foo:bar">
  <rng:ref name="__yang_metadata__"/>
  ...
</rng:element>
```

Note that it is not necessary to use such a reference for "rng:element" patterns corresponding to anyxml data nodes because they already permit any XML attributes to be attached to their instances.

The second step of the DSDL mapping procedure, i.e., the transformation of the hybrid schema to RELAX NG, Schematron and DSRL schemas, is unaffected by the inclusion of "md:annotation".

7. Metadata YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-yang-metadata@2015-09-17.yang"

module ietf-yang-metadata {

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

  prefix "md";

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

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

    WG Chair: Thomas Nadeau
```

<mailto:tnadeau@lucidvision.com>

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

WG Chair: Kent Watsen
<mailto:kwatsen@juniper.net>

Editor: Ladislav Lhotka
<mailto:lhotka@nic.cz>;

description

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

Copyright (c) 2015 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>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<http://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<http://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices."

```
revision 2015-09-17 {  
  description  
    "Initial revision."  
  reference  
    "RFC XXXX: Defining and Using Metadata with YANG";  
}
```

```
extension annotation {  
  argument name;  
  description  
    "This extension allows for defining metadata annotations in  
    YANG modules. The 'md:annotation' statement can appear only at  
    the top level of a YANG module.
```

The argument of the 'md:annotation' statement defines the name of the annotation. Syntactically it is a YANG identifier as defined in RFC 6020bis, sec. 6.2.

An annotation defined with this extension statement inherits the namespace and other context from the YANG module in which it is defined.

Data type of the annotation value is specified in the same way as for a leaf data node using the 'type' statement.

Semantics of the annotation and other documentation can be specified using the following standard YANG substatements (all are optional): 'description', 'if-feature', 'reference', 'status', and 'units'.

A server announces support for a particular annotation by including the module in which the annotation is defined among the advertised YANG modules (e.g., in NETCONF hello message or yang-library). The annotation then can be attached to any instance of data node defined in any YANG module that is advertised by the server.

XML and JSON encoding of annotations is defined in RFC XXXX."

```
}  
}
```

<CODE ENDS>

8. IANA Considerations

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

This document registers a URI in the "IETF XML registry" [RFC3688]. Following the format in RFC 3688, the following registration has been made.

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

Registrant Contact: The NETMOD WG of the IETF.

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-yang-metadata  
namespace:     urn:ietf:params:xml:ns:yang:ietf-yang-metadata  
prefix:        md  
reference:     RFC XXXX  
-----
```

9. Security Considerations

This document introduces a mechanism for defining metadata annotations in YANG modules and attaching them to instances of YANG data nodes. By itself, this mechanism represents no security threat. Security implications of a particular annotation defined using this mechanism MUST be duly considered and documented in the the annotation's definition.

An annotation SHOULD be subject to the same or stricter access control rules as the data node instance to which the annotation is attached.

10. Acknowledgments

The author wishes to thank Andy Bierman, Martin Bjorklund, Benoit Claise, Juergen Schoenwaelder, and Kent Watsen for their helpful comments and suggestions.

11. References

11.1. Normative References

- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", draft-ietf-netmod-rfc6020bis-06 (work in progress), July 2015.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-05 (work in progress), September 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6110] Lhotka, L., Ed., "Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content", RFC 6110, DOI 10.17487/RFC6110, February 2011, <<http://www.rfc-editor.org/info/rfc6110>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [W3C.REC-xml-infoset-20040204]
Cowan, J. and R. Tobin, "XML Information Set (Second Edition)", World Wide Web Consortium Recommendation REC-xml-infoset-20040204, February 2004, <<http://www.w3.org/TR/2004/REC-xml-infoset-20040204>>.
- [W3C.REC-xml-names11-20060816]
Bray, T., Hollander, D., Layman, A., and R. Tobin, "Namespaces in XML 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml-names11-20060816, August 2006, <<http://www.w3.org/TR/2006/REC-xml-names11-20060816>>.

11.2. Informative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.
- [ISO.19757-1]
International Organization for Standardization, "Document Schema Definition Languages (DSDL) - Part 1: Overview", ISO/IEC 19757-1, November 2004.

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

Appendix A. Change Log

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

A.1. Changes Between Revisions -01 and -02

- o The "type" statement became mandatory.
- o Terminology section was extended.
- o The annotation "inactive" defined in the example module was replaced with "last-modified" that is supposedly less controversial.
- o Introduction now states limitation due to XML attribute properties.
- o A recommendation was added to define annotations in a module by themselves.
- o Section "Using Annotations" was added.
- o An example for "anyxml" was added.
- o RFC 6241 was moved to informative references.

A.2. Changes Between Revisions -00 and -01

- o Define JSON encoding for annotations attached to 'anydata' nodes.

A.3. Changes Between draft-lhotka-netmod-yang-metadata-01 and draft-ietf-netmod-yang-metadata-00

- o References to RFC 6020 were changed to the 6020bis I-D.
- o Text about RFC 2119 key words was added to "ietf-yang-metadata" module description.

A.4. Changes Between draft-lhotka-netmod-yang-metadata-00 and -01

- o Encoding of annotations for anyxml nodes was changed to be the same as for leafs. This was necessary because anyxml value now needn't be an object.

- o It is stated that "md:annotation" statement defines only the syntax of an annotation.
- o Allowed "if-feature" as a substatement of "md:annotation".

Author's Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

K. D'Souza
AT&T
A. Shaikh
Google
R. Shakir
Jive Communications
October 19, 2015

Catalog and registry for YANG models
draft-openconfig-netmod-model-catalog-00

Abstract

This document presents an approach for a YANG model catalog and registry that allows users to find models relevant to their use cases from the large and growing number of YANG modules being published. The model catalog may also be used to define bundles of YANG modules required to realize a particular service or function.

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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. Model catalog and registry requirements	3
3. Model catalog schema	5
3.1. Module information	5
4. Module composition with bundles	7
4.1. Schema for module bundles	8
5. Security Considerations	8
6. IANA Considerations	8
7. YANG modules	9
8. References	22
8.1. Normative references	22
8.2. Informative references	23
Authors' Addresses	23

1. Introduction

As YANG [RFC6020] adoption and usage grows, the number of YANG models (and corresponding module and submodule files) published is increasing rapidly. This growing collection of modules potentially enables a large set of management use cases, but from a user perspective, it is a daunting task to navigate the largely ad-hoc landscape of models to determine their functionality, availability, and implementations. For example, the IETF Routing Area Coordination page [RTG-AD-YANG] currently tracks nearly 150 YANG models related to layer 2 and layer 3 technologies.

YANG models are also being developed and published beyond the IETF, for example by open source projects, other standards organizations, and industry forums. These efforts are generally independent from each other and often result in overlapping models. While we recognize that models may come from multiple sources, the current approach of having a flat online listing of models is not sufficient to help users find the models they need, along with the information to retrieve and utilize the models in actual operational systems. There is a need for a wider registry and catalog of available models that provides a central reference for model consumers and developers.

The idea of a model catalog is inspired by service catalogs in traditional IT environments. Service catalogs serve as software-based registries of available services with information needed to discover and invoke them.

In earlier proposals [I-D.openconfig-netmod-model-structure] we motivated the need for a common structure that allows a set of models to be used together coherently in order to manage, for example, a complete network device. Other efforts have subsequently proposed further options for modeling the complete device structure [I-D.rtgwangdt-rtgwg-device-model]. We also briefly described the notion of a model catalog to provide a structured view of all of the models available from different organizations. In this document, we further elaborate on some of the details and use cases for a model catalog and registry.

There are recent proposals that address related issues in terms of understanding the set of YANG models available on a device [I-D.ietf-netconf-yang-library], and how to classify models based on their role in describing a multi-layer service [I-D.bogdanovic-netmod-yang-model-classification]. The latter, in particular, describes a taxonomy for classifying YANG models that could also be used in the model catalog, though it does not address the problem of classifying model functionality, which is a key requirement.

2. Model catalog and registry requirements

At a high level, the model catalog must provide enough information for users to determine which models are available to describe a specific service or technology, and attributes of those models that would help the user select the best model for their scenario. While this draft does not specifically address selection criteria -- they would be specific to each user -- some examples include:

- o model maturity, including availability of server implementations (e.g., native device support)
- o available of co-requisite models, and complexity of the model dependencies
- o identity and reputation of the entity or organization publishing the model

The model catalog should, therefore, include key information about YANG modules, including:

- o organization responsible for publishing and maintaining the module with contact information; organizations may include standards bodies (SDOs), industry forums, open source projects, individuals, etc.

- o classification of the module or model, which could be along several axes, e.g., functional category, service vs. element models, commercial vs. free-to-use, etc.
- o for open models, the license under which the model is distributed; this is important if there are limitations on how the model may be modified or redistributed
- o module dependencies, e.g., a list of all of the YANG modules that are required
- o pointer to the YANG module, e.g., a URI that can be machine-processed
- o implementation information, for example, a list of available server implementations that support the module
- o authentication information to allow users to verify that the model they retrieve is authentic and unaltered

Establishing a globally applicable classification scheme for models is not straightforward -- each organization developing models likely has its own taxonomy or organization strategy for YANG modules. This is an area of the catalog that is likely to require extensibility and customization, e.g., by letting each organization augment the schema with its own categories. Similarly, users may want to define their own classifications for use by internal systems.

The proposed catalog schema should be useful as a local database, deployed by a single user, and also as a global registry that can be used to discover available models. For example, the local catalog could be used to define the approved set of models for use within an organization, while the registry serves as a channel for all model developers to make information about their models available. The IETF XML Registry [RFC3688], maintained by IANA serves a similar purpose for XML documents used in IETF protocols, but it is limited to IETF-defined YANG models, is tied to XML encoded data, and has a very limited schema.

The registry implementation could be as simple as a metadata database that reflects the proposed catalog schema, along with means for online access and viewing. A key requirement for the online registry would be a robust query capability that allows users to search for modules meeting a variety of selection criteria, along with an easy way to retrieve modules (where applicable).

3. Model catalog schema

We propose a schema for the model catalog defined using YANG (see the modules in Section 7). The YANG modules in the catalog are organized at the top level by the publishing organization and its associated contact information. The catalog structure is shown below.

```
+--rw organizations
  +--rw organization* [name]
    +--rw name          string
    +--rw type?         identityref
    +--rw contact?      string
    +--rw modules
      +--rw module* [name]
        +--rw name          string
        +--rw namespace?   string
        +--rw prefix?      string
        +--rw revision?    string
        +--rw summary?     string
        +--rw module-version? string
        +--rw module-hierarchy
        |   ...
        +--rw classification
        |   ...
        +--rw dependencies
        |   ...
        +--rw module-usage
        |   ...
        +--rw implementations
        |   ...
```

In this model, each organization publishes a list of available modules, each module having associated data describing its classification, dependencies, usage information, and implementation information. In addition, some of the basic module metadata is included in the catalog, e.g., namespace, prefix, and revision.

3.1. Module information

Each module has several types of information associated with it. These are described below.

The basic information includes module metadata as mentioned above and also the location of module in its own dependency chain. The module-hierarchy container indicates whether the module is a submodule of another module, and has a reference to its parent module.

The classification data is meant to capture some base information but leave the taxonomy largely to model publishers. The category and subcategory leaves are identities that are expected to be augmented with additional values. The classification also includes a status to indicate the development or deployment status of the module, e.g., whether it is purely experimental, or mature enough for production use. The classification data is shown below:

```
+++rw module* [name]
  +-rw classification
    +-rw status?      identityref
    +-rw category?    identityref
    +-rw subcategory? identityref
```

In this initial version of the catalog schema, the module dependencies are represented as a simple list of references to co-requisite modules. The model assumes that required modules are also represented in the catalog, and that only the first-level dependencies are included in the list. That is, each of the listed modules can be examined to determine its dependencies.

The usage data contains information required to retrieve and validate the module. Specifically, it includes authentication and validation data to ensure the origin and integrity of the module, respectively. The authentication information will be further developed in future revisions of the document; in the current version, these can be considered placeholders. This section also includes a URI for modules that can be downloaded directly. This part of the schema is shown below:

```
+++rw module* [name]
  +-rw module-usage
    +-rw authentication? string
    +-rw md5-hash?       string
    +-rw access-uri?     inet:uri
```

The implementation container provides information about known implementations of the module, for example by network devices or other servers. This data is structured as a list to account for multiple implementations of a module, e.g., by different vendors. It includes some basic information about the platform on which the module is supported, and the status of the implementation, but it is expected that details and limitations of the implementation will require consulting the implementor. The implementation information in the catalog is shown below:

```
    +--rw module* [name]
      +--rw implementations
        +--rw implementation* [implementation-id]
          +--rw implementation-id      string
          +--rw description?           string
          +--rw reference?              union
          +--rw implementor-name?       string
          +--rw platform?               string
          +--rw platform-version?       string
          +--rw implementation-status?  identityref
```

4. Module composition with bundles

From an operational perspective, the utility of a single module is quite limited. Most, if not all, use cases require multiple modules that work together coherently. Managing a network device typically requires configuration and operational state models for device-wide services, network protocols, virtual instances, etc. Network services, such as those delivered by many service providers, require not only infrastructure-level management models, such as devices and protocols, but also service-level models that describe service parameters.

The model catalog and registry provides a common way to define service bundles, or recipes, that describe the set of modules required for realizing the feature or service. For example, a Layer 3 VPN bundle would list its required configuration and state models, including VRFs, interfaces, BGP, policy, ACLs, and QoS. Similar bundles can be defined for other services or use cases, for example, basic Internet operations such as adding new peers or customers, or setting up Layer 2 VPNs. Note these bundle definitions complement actual configuration models for such services, which may focus on providing an abstracted set of configuration or operational state variables. These variables would then be mapped onto device level variables. We leave discussion of such mapping mechanisms to future revisions.

Bundle definitions are particularly useful for organizations that identify and validate a set of models that are used to build a service, and then define an approved bundle based on that set. Users within the organization can be assured that the corresponding bundles are known to work together to support the desired service. Another use case for bundle definitions is for third-party testing or certification organizations to provide services to validate a set of modules and maintain the bundle.

4.1. Schema for module bundles

We propose an initial YANG-defined schema for describing a service bundle for building composite services and functions (shown below).

```

+--rw bundle
  +--rw name?          string
  +--rw version?       string
  +--rw description?   string
  +--rw category?     string
  +--rw subcategory?   string
  +--rw modules
    +--rw module* [module-type]
      +--rw module-type          string
      +--rw catalog-reference? -> /cat:organizations/.../module/name
      +--rw application-sequence? uint8

```

Each bundle includes basic information such as the name of the feature or service, the bundle version, and its category and subcategory. The modules comprising the bundle are contained in the modules list with a reference to the module in the catalog. The application sequence number can be used to indicate an ordering of the modules in realizing the service, for example, device or element configuration modules followed by service configuration models. The application sequence is a high level indication; a complete realization of the service would require a detailed definition of the mapping between module variables at different levels as discussed in Section 4.

5. Security Considerations

The model catalog and registry described in this document do not define actual configuration and state data, hence are not directly responsible for security risks.

However, since the model catalog is intended to be an authoritative and authenticated database of published modules, there are security considerations in securing the catalog (both contents and access), and also in authenticating organizations that deposit data into the catalog.

6. IANA Considerations

The YANG model catalog is intended to complement the IANA XML Registry. YANG modules defined in this document may be entered in the XML registry if they are placed or redirected for the standards track, with an appropriate namespace URI.

7. YANG modules

The model catalog and associated types modules are listed below.

```
<CODE BEGINS> file openconfig-module-catalog.yang
module openconfig-module-catalog {

    yang-version "1";

    // namespace
    namespace "http://openconfig.net/yang/module-catalog";

    prefix "cat";

    // import some basic types
    import ietf-inet-types { prefix inet; }
    import openconfig-catalog-types { prefix cat-types; }

    // meta
    organization "OpenConfig working group";

    contact
        "OpenConfig working group
        www.openconfig.net";

    description
        "This module provides a schema for cataloging and describing
        YANG models published across various organizations.";

    revision "2015-10-18" {
        description
            "Initial revision";
        reference "TBD";
    }

    // extension statements

    // feature statements

    // identity statements

    // typedef statements

    // grouping statements

    grouping module-implementation-information {
        description
```

```
"Data describing any available implementations";

container implementations {
  description
    "Container for module implementation information";

  list implementation {
    key implementation-id;
    description
      "List of available implementations, keyed by an identifier
      provided by either the implementor or the module
      maintainer. Such a key avoids needing a complex composite
      key to uniquely identify an implementation.";

    leaf implementation-id {
      type string;
      description
        "An identifier for the implementation, provided by the
        implementor or the module maintainer. This id should
        uniquely identify a specific implementation of the
        module, e.g., based on the vendor, platform, and platform
        version.";
    }

    leaf description {
      type string;
      description
        "A text summary of important information about the
        implementation";
    }

    leaf reference {
      type union {
        type string;
        type inet:uri;
      }
      description
        "A URI or text reference to more detailed information
        about the implementation.";
    }

    leaf implementor-name {
      type string;
      description
        "Name of the vendor or entity providing the module
        implementation";
    }
  }
}
```

```
    leaf platform {
        type string;
        description
            "Name of the server platform on which the implementation
            is available -- this could be the model name of a network
            device, a server OS, etc.";
    }

    leaf platform-version {
        type string;
        description
            "Implementor-defined version name or number for the
            module implementation, corresponding to the platform.
            This could be the firmware version of a network device
            such as a router, OS version, or other server platform
            version.";
    }

    leaf implementation-status {
        type identityref {
            base cat-types:implementation-status-type;
        }
        description
            "Indicates the status of the implementation, e.g.,
            complete, partial, in-progress, etc. Implementors
            may define additional values for the base identity";
    }
}

}

}

grouping module-dependency-information {
    description
        "Information about module dependencies";

    container dependencies {
        description
            "Container for information about module dependencies";

        leaf-list required-module {
            type leafref {
                path "../..../name";
            }
            description
                "//TODO: should this list be complete, or only the first-
                //level dependencies?
                "A simple list of modules that are prerequisites for the
                current module. It is expected that each of the required
```

```
        modules would in turn list their dependencies.  The list
        values should be references to other modules in the
        catalog.";
    }
}

grouping module-classification-information {
  description
    "Data describing the module's classification(s)";

  container classification {
    description
      "Container for data describing the module's classification";

    leaf status {
      type identityref {
        base cat-types:module-status-type;
      }
      description
        "Status of the module -- experimental, standards-track,
        production, etc.";
    }

    leaf category {
      type identityref {
        base cat-types:module-category-base;
      }
      description
        "Categorization of the module based on identities defined
        by the publishing organizations.";
    }

    leaf subcategory {
      type identityref {
        base cat-types:module-subcategory-base;
      }
      description
        "Sub-categorization of the module based on identities
        defined by the publishing organizations.";
    }
  }
}

grouping module-usage-information {
  description
    "Data pertaining to retrieval and usage of the module";
```

```
    container module-usage {
      description
        "Container for data pertaining to retrieval and usage of the
        module";

      leaf authentication {
        //TODO: requires more detailed model for different types
        //of authentication / validation schemes
        type string;
        description
          "Authentication information to allow
          users to verify that the model originates from
          stated organization, e.g., X.509 certificate";
      }

      leaf md5-hash {
        type string;
        description
          "MD5 hash of the module file, used by users to validate
          data integrity";
      }

      leaf access-uri {
        type inet:uri;
        description
          "URI where module can be downloaded.  Modules may be
          made available from the catalog maintainer, or directly
          from the publisher";
      }
    }
  }

  grouping module-base-information {
    description
      "Basic information describing the module, e.g., the
      YANG metadata in the module preface.";

    leaf name {
      type string;
      description
        "The module name, as defined in the YANG module file.";
    }

    leaf namespace {
      //type inet:uri;
      type string;
      description
        "Published namespace of module";
    }
  }
}
```

```
    }

    leaf prefix {
        type string;
        description "Published prefix of module";
    }

    leaf revision {
        type string;
        description
            "Date in the revision statement of the module";
    }

    leaf summary {
        type string;
        description
            "Brief summary of the module description";
    }

    leaf module-version {
        type string;
        description
            "Optional version number for the module, in addition to the
            YANG revision statement";
    }

    container module-hierarchy {

        description
            "YANG module hierarchy specification";

        leaf module-hierarchy-level {
            type uint8 {
                range 1..5;
            }
            default 1;
            description
                "Module hierarchy level. If this is a sub-module,
                it is set to > 1, depending
                on the hierarchy level of the sub-module";
        }

        leaf module-parent {
            when "../module-hierarchy-level > '1'" {
                description "Only applicable to sub-modules";
            }
            type leafref {
                path "../../name";
            }
        }
    }
}
```

```
        }
        description
            "Parent module, if this is a sub-module";
    }
} //module-base-information

grouping organization-information {
    description
        "Data describing the publisher of the module";

    leaf name {
        type string;
        description
            "Name of Organization defining YANG Module:"
            + "Standards Body examples:"
            + "    ietf, ieee, opendaylight, etc."
            + "Commercial entity examples:"
            + "    AT&T"
            + "Name of industry forum examples:"
            + "    openconfig, other";
    }

    leaf type {
        type identityref {
            base cat-types:organization-type;
        }
        description
            "YANG modules publication authority";
    }

    leaf contact {
        type string;
        description
            "Contact information for the publishing organization";
    }
}

grouping module-catalog-top {
    description
        "Top level structure of the module catalog";

    container organizations {

        description
            "List of organizations owning modules";
    }
}
```

```
list organization {
    key "name";

    description
        "List of organizations defining the YANG Modules";

    uses organization-information;

    container modules {
        description
            "Modules published by this organization";

        list module {
            key name;
            description
                "List of published modules from the organization";

            uses module-base-information;
            uses module-classification-information;
            uses module-dependency-information;
            uses module-usage-information;
            uses module-implementation-information;
        }
    }
}

// data definition statements

uses module-catalog-top;

// augment statements

// rpc statements

// notification statements
}
<CODE ENDS>

<CODE BEGINS> file openconfig-catalog-types.yang
module openconfig-catalog-types {

    yang-version "1";
```



```
// namespace
namespace "http://openconfig.net/yang/catalog-types";

prefix "cat-types";

// import some basic types

// meta
organization "OpenConfig working group";

contact
  "OpenConfig working group
  www.openconfig.net";

description
  "This module defines types and identities used by the OpenConfig
  YANG module catalog model.";

revision "2015-10-18" {
  description
    "Initial revision";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

identity implementation-status-type {
  description
    "Indications of the status of a module's implementation on a
    device or server";
}

identity in-progress {
  base implementation-status-type;
  description
    "Implementation is in progress";
}

identity planned {
  base implementation-status-type;
  description
    "Implementation is planned";
}
```

```
identity complete {
  base implementation-status-type;
  description
    "Implementation is complete and fully supports the model";
}

identity partial {
  base implementation-status-type;
  description
    "Implementation is complete, but only supports the model
    partially";
}

identity module-status-type {
  description
    "Indicates the deployment status of the module";
}

identity experimental {
  base module-status-type;
  description
    "Module should be considered experimental, not deployed in
    production settings";
}

identity production {
  base module-status-type;
  description
    "Module is suitable for use in production, or has been
    deployed in production";
}

identity module-category-base {
  description
    "Base identity for the module category. It is expected that
    publishing organizations will define additional derived
    identities to describe their categorization scheme.";
}

identity module-subcategory-base {
  description
    "Base identity for the module subcategory. It is expected that
    publishing organizations will define additional derived
    identities to describe their categorization scheme.";
}

identity organization-type {
  description
```

```
        "Publishing organization type for the set of modules";
    }

    identity standards {
        base organization-type;
        description
            "Standards development organization (SDO) publisher type";
    }

    identity industry {
        base organization-type;
        description
            "Industry forum or other industry group";
    }

    identity commercial {
        base organization-type;
        description
            "Commercial entity, company, etc.";
    }

    identity individual {
        base organization-type;
        description
            "For modules published by an individual";
    }

    // typedef statements

    typedef yang-model-publisher
    {
        type enumeration {
            enum "standards";
            enum "commercial";
            enum "forum";
        }
    }

    // grouping statements

    // data definition statements

    // augment statements

    // rpc statements

    // notification statements
```

```
}  
<CODE ENDS>
```

The feature bundle module is listed below.

```
<CODE BEGINS> file openconfig-feature-bundle.yang  
module openconfig-feature-bundle {  
  
    // namespace  
    // TODO: change to an ietf or other more generic namespace  
    namespace "http://openconfig.net/yang/feature-bundle";  
  
    prefix bundle;  
  
    import openconfig-module-catalog { prefix cat; }  
  
    // meta  
    organization "OpenConfig working group";  
  
    contact  
        "OpenConfig working group  
        netopenconfig@googlegroups.com";  
  
    description  
        "This module can be used to build network features using  
        published YANG Models.";  
  
    revision "2015-10-18" {  
        description  
            "Initial Revision";  
        reference "TBD";  
    }  
  
    grouping bundle-information {  
  
        description  
            "Template defining the bundle";  
  
        leaf name {  
            type string;  
            description "Published name of bundle, for example:  
                l3vpn, l2vpn, internet-access";  
        }  
  
        leaf version {  
            type string;  
            description "bundle version number";  
        }  
    }  
}
```

```
    }

    leaf description {
        type string;
        description "User defined information about bundle";
    }

    leaf category {
        type string;
        description
            "Categorization of bundle such as:
             network, service, oam, experimental, other";
    }

    leaf subcategory {
        type string;
        description
            "Sub-Categorization of bundle such as:
             protocol, operational, other";
    }
} //bundle-template

grouping bundle-ingredients {

    description "Module ingredients used in bundle";

    container modules {

        description
            "Modules that comprise the bundle";

        list module {

            key "module-type";

            description
                "List of modules from yang-module-catalog comprising
                 the bundle";

            leaf module-type {
                type string;
                description
                    "A user-define type of the module";
            }

            leaf catalog-reference {
                type leafref {
```

```
        path "/cat:organizations"
            + "/cat:organization"
            + "/cat:modules"
            + "/cat:module"
            + "/cat:name";
    }
    description
        "Link to a module defined by a standards body";
}

leaf application-sequence {
    type uint8;
    description
        "Sequence number indicating order of application of
        module";
}
} //module-info
} //bundle-modules
} //bundle-ingredients

container bundle {

    description
        "Build the bundle";

    uses bundle-information;
    uses bundle-ingredients;

} //bundle
}
<CODE ENDS>
```

8. References

8.1. Normative references

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.

8.2. Informative references

[RTG-AD-YANG]

Wu, Q. and D. Sinicrope, "Routing Area Yang Coordinator's Summary Page", October 2015,
<<http://trac.tools.ietf.org/area/rtg/trac/wiki/RtgYangCoordSummary>>.

[I-D.openconfig-netmod-model-structure]

Shaikh, A., Shakir, R., D'Souza, K., and L. Fang,
"Operational Structure and Organization of YANG Models",
draft-openconfig-netmod-model-structure-00 (work in
progress), March 2015.

[I-D.rtgyangdt-rtgwg-device-model]

Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps,
"Network Device YANG Organizational Model", draft-
rtgyangdt-rtgwg-device-model-01 (work in progress),
September 2015.

[I-D.ietf-netconf-yang-library]

Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module
Library", draft-ietf-netconf-yang-library-02 (work in
progress), October 2015.

[I-D.bogdanovic-netmod-yang-model-classification]

Bogdanovic, D., Claise, B., and C. Moberg, "YANG Model
Classification", draft-bogdanovic-netmod-yang-model-
classification-05 (work in progress), October 2015.

Authors' Addresses

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Rob Shakir
Jive Communications, Inc.
1275 West 1600 North, Suite 100
Orem, UT 84057

Email: rjs@rob.sh

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

R. Wilton, Ed.
D. Ball
Cisco Systems
T. Singh
S. Sivaraj
Juniper Networks
October 19, 2015

Common Interface Extension YANG Data Models
draft-wilton-netmod-intf-ext-yang-01

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU. These properties are common to many types of interfaces on network routers and switches and are implemented by multiple network equipment vendors with similar semantics, even though some of the features are not formally defined in any published standard.

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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. Terminology	3
1.2. Tree Diagrams	3
2. Objectives	4
3. Interfaces Extensions Module	4
3.1. Bandwidth	5
3.2. Carrier Delay	6
3.3. Dampening	7
3.3.1. Suppress Threshold	7
3.3.2. Half-Life Period	8
3.3.3. Reuse Threshold	8
3.3.4. Maximum Suppress Time	8
3.4. Encapsulation	8
3.5. Loopback	8
3.6. MTU	8
3.7. Sub-interface	9
3.8. Transport Layer	10
4. Interfaces Ethernet-Like Module	10
5. Interfaces Common YANG Module	10
6. Interfaces Ethernet-Like YANG Module	19
7. Acknowledgements	21
8. IANA Considerations	21
9. Security Considerations	22
9.1. interfaces-common.yang	22
9.2. interfaces-ethernet-like.yang	23
10. References	23
10.1. Normative References	24
10.2. Informative References	24
Authors' Addresses	24

1. Introduction

This document defines two YANG RFC 6020 [RFC6020] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model defined in RFC 7223 [RFC7223] to support configuration of lower layer interface properties that are common across many types of network interface.

One of the aims of this draft is to provide a standard namespace and path for these configuration items regardless of the underlying interface type. For example a standard namespace and path for configuring or reading the MAC address associated with an interface is provided that can be used for any interface type that uses Ethernet framing.

Several of the augmentations defined here are not backed by any formal standard specification. Instead, they are for features that are commonly implemented in equivalent ways by multiple independent network equipment vendors. The aim of this draft is to define common paths and leaves for the configuration of these equivalent features in a uniform way, making it easier for users of the YANG model to access these features in a vendor independent way. Where necessary, a description of the expected behavior is also provided with the aim of ensuring vendors implementations are consistent with the specified behaviour.

Given that the modules contain a collection of discrete features with the common theme that they generically apply to interfaces, it is plausible that not all implementors of the YANG module will decide to support all features. Hence separate feature keywords are defined for each logically discrete feature to allow implementors the flexibility to choose which specific parts of the model they support.

The augmentations are split into two separate YANG modules that each focus on a particular area of functionality. The two YANG modules defined in this internet draft are:

interface-extensions.yang - Defines extensions to the IETF interface data model to support common configuration data nodes.

etherlike-interfaces.yang - Defines a module for any configuration and operational data nodes that are common across interfaces that use Ethernet framing.

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 RFC 2119 [RFC2119].

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" means state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, 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. Objectives

The aim of the YANG modules contained in this draft is to provide standard definitions for common interface based configuration on network devices.

The expectation is that the YANG leaves that are being defined are fairly widely implemented by network vendors. However, the features described here are mostly not backed by formal standards because they are fairly basic in their behavior and do not need to interoperate with other devices. Where required a concise explanation of the expected behavior is also provided to ensure consistency between vendors.

3. Interfaces Extensions Module

The Interfaces Common module provides some basic extensions to the IETF interfaces YANG module.

The module provides:

- o A bandwidth configuration leaf to specify the bandwidth available on an interface to control routing metrics.
- o A carrier delay feature used to provide control over short lived link state flaps.
- o An interface link state dampening feature that is used to provide control over longer lived link state flaps.
- o An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.

- o A loopback configuration leaf that is primarily aimed at loopback at the physical layer.
- o MTU configuration leaves applicable to all packet/frame based interfaces.
- o A transport layer leaf to indicate whether the interface handles traffic at L1, L2 or L3.
- o A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "interface-extensions" YANG module has the following structure:

```
module: interfaces-common
augment /if:interfaces/if:interface:
  +--rw bandwidth?    uint64
augment /if:interfaces/if:interface:
  +--rw carrier-delay
    +--rw down?      uint32
    +--rw up?        uint32
augment /if:interfaces/if:interface:
  +--rw dampening!
    +--rw half-life?      uint32
    +--rw reuse?          uint32
    +--rw suppress?       uint32
    +--rw max-suppress-time? uint32
augment /if:interfaces/if:interface:
  +--rw encapsulation
    +--rw (encaps-type)?
augment /if:interfaces/if:interface:
  +--rw loopback?    identityref
augment /if:interfaces/if:interface:
  +--rw l2-mtu?      uint16 {configurable-l2-mtu}?
augment /if:interfaces/if:interface:
  +--rw parent-interface? if:interface-ref
augment /if:interfaces/if:interface:
  +--rw transport-layer? enumeration
```

3.1. Bandwidth

The bandwidth configuration leaf allows the specified bandwidth of an interface to be reduced from the inherent interface bandwidth. The bandwidth leaf affects the routing metric cost associated with the interface.

Note that the bandwidth leaf does not actually limit the amount of traffic that can be sent/received over the interface. If required,

interface traffic can be limited to the required bandwidth by configuring an explicit QoS policy.

Note for reviewers: Given that the bandwidth only controls routing metrics, it may be more appropriate for this leaf, or an equivalent, to be defined as part of one of the routing YANG modules. Although conversely, it is also worth considering that the corresponding existing CLI configuration command is an interface level bandwidth command in many implementations.

3.2. Carrier Delay

The carrier delay feature augments the IETF interfaces data model with configuration for a simple algorithm that is used, generally on physical interfaces, to suppress short transient changes in the interface link state. It can be used in conjunction with the dampening feature described in Section 3.3 to provide effective control of unstable links and unwanted state transitions.

The principal of the carrier delay feature is to use a short per interface timer to ensure that any interface link state transition that occurs and reverts back within the specified time interval is entirely suppressed without providing any signalling to any upper layer protocols that the state transition has occurred. E.g. in the case that the link state transition is suppressed then there is no change of the `/if:interfaces-state/if:interface/oper-status` or `/if:interfaces-state/if:interfaces/last-change` leaves for the interface that the feature is operating on. One obvious side effect of using this feature that is worth noting is that any state transition will always be delayed by the specified time interval.

The configuration allows for separate timer values to be used in the suppression of down->up->down link transitions vs up->down->up link transitions.

The carrier delay down timer leaf specifies the amount of time that an interface that is currently in link up state must be continuously down before the down state change is reported to higher level protocols. Use of this timer can cause traffic to be black holed for the configured value and delay reconvergence after link failures, therefore its use is normally restricted to cases where it is necessary to allow enough time for another protection mechanism (such as an optical layer automatic protection system) to take effect.

The carrier delay up timer leaf specifies the amount of time that an interface that is currently in link down state must be continuously up before the down->up link state transition is reported to higher level protocols. This timer is generally useful as a debounce

mechanism to ensure that a link is relatively stable before being brought into service. It can also be used effectively to limit the frequency at which link state transition events can occur. The default value for this leaf is determined by the underlying network device.

3.3. Dampening

The dampening feature introduces a configurable exponential decay mechanism to suppress the effects of excessive interface link state flapping. This feature allows the network operator to configure a device to automatically identify and selectively dampen a local interface which is flapping. Dampening an interface keeps the interface operationally down until the interface stops flapping and becomes stable. Configuring the dampening feature can improve convergence times and stability throughout the network by isolating failures so that disturbances are not propagated, which reduces the utilization of system processing resources by other devices in the network and improves overall network stability.

The basic algorithm uses a counter that is nominally increased by 1000 units every time the underlying interface link state changes from up to down. If the counter increases above the suppress threshold then the interface is kept down (and out of service) until either the maximum suppression time is reached, or the counter has reduced below the reuse threshold. The half-life period determines that rate at which the counter is periodically reduced. Implementations are not required to use a penalty of 1000 units in their dampening algorithm, but should ensure that the Suppress Threshold and Reuse Threshold values are scaled relative to the nominal 1000 unit penalty to ensure that the same configuration values provide consistent behaviour. The configurable values are described in more detail below.

3.3.1. Suppress Threshold

The suppress threshold is the value of the accumulated penalty that triggers the device to dampen a flapping interface. The flapping interface is identified by the device and assigned a penalty for each up to down link state change, but the interface is not automatically dampened. The device tracks the penalties that a flapping interface accumulates. When the accumulated penalty reaches the default or configured suppress threshold, the interface is placed in a dampened state.

3.3.2. Half-Life Period

The half-life period determines how fast the accumulated penalties can decay exponentially. Any penalties that have been accumulated on a flapping interface are reduced by half after each half-life period.

3.3.3. Reuse Threshold

If, after one or more half-life periods, the accumulated penalty decreases below the reuse threshold and the underlying interface link state is up then the interface is taken out of dampened state and allowed to go up.

3.3.4. Maximum Suppress Time

The maximum suppress time represents the maximum amount of time an interface can remain dampened when a penalty is assigned to an interface. The default of the maximum suppress timer is four times the half-life period. The maximum value of the accumulated penalty is calculated using the maximum suppress time, reuse threshold and half-life period.

3.4. Encapsulation

The encapsulation container holds a choice node that is to be augmented with datalink layer specific encapsulations, such as HDLC, PPP, or sub-interface 802.1Q tag match encapsulations. It ensures that an interface can only have a single datalink layer protocol configured.

3.5. Loopback

The loopback configuration leaf allows any physical interface to be configured to be in one of the possible following physical loopback modes, i.e. internal loopback, line loopback, or use of an external loopback connector. The use of YANG identities allows for the model to be extended with other modes of loopback if required.

3.6. MTU

Two MTU configuration leaves are provided to program the layer 2 interface in two different ways. Different mechanisms are provided to reflect the fact that devices handle their MTU configuration in different ways. A given device would only normally be expected to support MTU configuration using one of these mechanisms.

The preferable way to configure MTU is using the l2-mtu leaf that specifies the maximum size of a layer 2 frame including header and

payload, but excluding any frame checksum (FCS) bytes. The payload MTU available to higher layer protocols is calculated from the l2-mtu after taking the layer 2 header size into account.

For Ethernet interfaces carrying 802.1Q VLAN tagged frames, the l2-mtu excludes the 4-8 byte overhead of any known (e.g. explicitly matched by a child sub-interface) 801.1Q VLAN tags.

The alternative way to configure MTU is using the l3-mtu leaf that specifies the maximum size of payload carried by a layer 2 frame. The maximum size of the layer 2 frame can then be derived by adding on the size of the layer 2 header overheads.

Note for reviewers: Is it correct/beneficial to support l3-mtu? It would be easier for clients if they only had a single MTU that they could configure. Can all devices sensibly handle an l2-mtu configuration leaf?

3.7. Sub-interface

The sub-interface feature specifies the minimal leaves required to define a child interface that is parented to another interface.

A sub-interface is a logical interface that handles a subset of the traffic on the parent interface. Separate configuration leaves are used to classify the subset of ingress traffic received on the parent interface to be processed in the context of a given sub-interface. All egress traffic processed on a sub-interface is given to the parent interface for transmission. Otherwise, a sub-interface is like any other interface in /if:interfaces and supports the standard interface features and configuration.

For some vendor specific interface naming conventions the name of the child interface is sufficient to determine the parent interface, which implies that the child interface can never be reparented to a different parent interface after it has been created without deleting the existing the sub-interface and recreating a new sub-interface. Even in this case it is useful to have a well defined leaf to cleanly identify the parent interface.

The model also allows for arbitrarily named sub-interfaces by having an explicit parent-interface leaf define the child -> parent relationship. In this naming scenario it is also possible for implementations to allow for logical interfaces to be reparented to new parent interfaces without needing the sub-interface to be destroyed and recreated.

3.8. Transport Layer

The transport layer leaf provides additional information as to which layer an interface is logically operating and forwarding traffic at. The implication of this leaf is that for traffic forwarded at a given layer that any headers for lower layers are stripped off before the packet is forwarded at the given layer. Conversely, on egress any lower layer headers must be added to the packet before it is transmitted out of the interface.

This leaf can also be used as a simple mechanism to determine whether particular types of configuration are valid. E.g. a layer 2 QoS policy could ensure that it is only applied to a interface running at transport layer 2.

4. Interfaces Ethernet-Like Module

The Interfaces Ethernet-Like Module is a small module that contains all configuration and operational data that is common across interface types that use Ethernet framing as their datalink layer encapsulation.

This module currently contains leaves for the configuration and reporting of the operational MAC address and the burnt-in MAC address (BIA) associated with any interface using Ethernet framing.

The "interfaces-ethernet-like" YANG module has the following structure:

```
module: interfaces-ethernet-like
augment /if:interfaces/if:interface:
  +--rw ethernet-like
    +--rw mac-address?   yang:mac-address
augment /if:interfaces-state/if:interface:
  +--ro ethernet-like
    +--ro mac-address?   yang:mac-address
    +--ro bia-mac-address? yang:mac-address
```

5. Interfaces Common YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223].

```
<CODE BEGINS> file "interfaces-common@2015-10-19.yang"
module interfaces-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:interfaces-common";
  prefix if-cmn;
```

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

import iana-if-type {
  prefix ianaift;
}

organization
  "Cisco Systems, Inc.
   Customer Service

   Postal: 170 W Tasman Drive
   San Jose, CA 95134

   Tel: +1 1800 553-NETS

   E-mail: cs-yang@cisco.com";

contact
  "Robert Wilton - rwilton@cisco.com";

description
  "This module contains common definitions for extending the IETF
   interface YANG model (RFC 7223) with common configurable layer 2
   properties.";

revision 2015-10-19 {
  description
    "Add support for various common interface configuration
     parameters that are likely to be widely implemented by various
     network device vendors.";

  reference "Internet draft: draft-wilton-netmod-intf-ext-yang-01";
}

feature bandwidth {
  description
    "This feature indicates that the device supports configurable
     interface bandwidth.";
  reference "Section 3.1 Bandwidth";
}

feature carrier-delay {
  description
    "This feature indicates that configurable interface
     carrier delay is supported, which is a feature is used to
     limit the propagation of very short interface link state
```

```
        flaps.";
    reference "Section 3.2 Carrier Delay";
}

feature dampening {
    description
        "This feature indicates that the device supports interface
        dampening, which is a feature that is used to limit the
        propagation of interface link state flaps over longer
        periods";
    reference "Section 3.3 Dampening";
}

feature loopback {
    description
        "This feature indicates that configurable interface loopback
        is supported.";
    reference "Section 3.5 Loopback";
}

feature configurable-l2-mtu {
    description
        "This feature indicates that the device supports configuring
        layer 2 MTUs on interfaces. Such MTU configurations include
        the layer 2 header overheads (but exclude any FCS overhead).
        The payload MTU available to higher layer protocols is either
        derived from the layer 2 MTU, taking into account the size of
        the layer 2 header, or is further restricted by explicit layer
        3 or protocol specific MTU configuration.";
    reference "Section 3.6 MTU";
}

feature sub-interfaces {
    description
        "This feature indicates that the device supports the
        instantiation of sub-interfaces. Sub-interfaces are defined
        as logical child interfaces that allow features and forwarding
        decisions to be applied to a subset of the traffic processed
        on the specified parent interface.";
    reference "Section 3.7 Sub-interface";
}

feature transport-layer {
    description
        "This feature indicates that the device supports configurable
        transport layer.";
    reference "Section 3.8 Transport Layer";
}
```

```
/*
 * Define common identities to help allow interface types to be
 * assigned properties.
 */
identity sub-interface {
    description "Base type for generic sub-interfaces. New or custom
        interface types can derive from this type to
        inherit generic sub-interface configuration";
}

identity ethSubInterface{
    base ianaift:l2vlan;
    base sub-interface;

    description "Sub-interface of any interface types that uses
        Ethernet framing (with or without 802.1Q tagging)";
}

identity loopback {
    description "Base type for interface loopback options";
}
identity loopback-internal {
    base loopback;
    description "All egress traffic on the interface is internally
        looped back within the interface to be received on
        the ingress path.";
}
identity loopback-line {
    base loopback;
    description "All ingress traffic received on the interface is
        internally looped back within the interface to the
        egress path.";
}
identity loopback-connector {
    base loopback;
    description "The interface has a physical loopback connector
        attached to that loops all egress traffic back into
        the interface's ingress path, with equivalent
        semantics to loopback-internal";
}

/*
 * Augments the IETF interfaces model with a leaf to explicitly
 * specify the bandwidth available on an interface.
 */
augment "/if:interfaces/if:interface" {
    if-feature "bandwidth";
```

```
description "Add a top level node for interface bandwidth.";
leaf bandwidth {
    type uint64;
    units kbps;
    description
        "The bandwidth available on the interface in Kb/s. This
        configuration is used by routing protocols to adjust the
        metrics associated with the interface, but does not limit
        the amount of traffic that can be sent or received on the
        interface. A separate QoS policy would need to be configured
        to limit the ingress or egress traffic. If not configured,
        the default bandwidth is the maximum available bandwidth of
        the underlying interface.";
}
}

/*
 * Defines standard YANG for the Carrier Delay feature.
 */
augment "/if:interfaces/if:interface" {
    if-feature "carrier-delay";
    description "Augments the IETF interface model with
        carrier delay configuration for interfaces that
        support it.";

    container carrier-delay {
        description "Holds carrier delay related feature
            configuration";
        leaf down {
            type uint32;
            units milliseconds;
            description
                "Delays the propagation of a 'loss of carrier signal' event
                that would cause the interface state to go down, i.e. the
                command allows short link flaps to be suppressed. The
                configured value indicates the minimum time interval (in
                milliseconds) that the carrier signal must be continuously
                down before the interface state is brought down. If not
                configured, the behaviour on loss of carrier signal is
                vendor/interface specific, but with the general
                expectation that there should be little or no delay.";
        }
        leaf up {
            type uint32;
            units milliseconds;
            description
                "Defines the minimum time interval (in milliseconds) that
                the carrier signal must be continuously present and
```

```
error free before the interface state is allowed to
transition from down to up.  If not configured, the
behaviour is vendor/interface specific, but with the
general expectation that sufficient default delay
should be used to ensure that the interface is stable
when enabled before being reported as being up.
Configured values that are too low for the hardware
capabilities may be rejected.";
    }
}
}

/*
 * Augments the IETF interfaces model with a container to hold
 * generic interface dampening
 */
augment "/if:interfaces/if:interface" {
  if-feature "dampening";
  description
    "Add a container for interface dampening configuration";

  container dampening {
    presence "Enable interface link flap dampening with default
      settings (that are vendor/device specific)";
    description "Interface dampening limits the propagation of
      interface link state flaps over longer periods";
    leaf half-life {
      type uint32;
      units seconds;
      description
        "The Time (in seconds) after which a penalty reaches half
        its original value. Once the interface has been assigned
        a penalty, the penalty is decreased by half after the
        half-life period. For some devices, the allowed values may
        be restricted to particular multiples of seconds. The
        default value is vendor/device specific.";
    }

    leaf reuse {
      type uint32;
      description
        "Penalty value below which a stable interface is
        unsuppressed (i.e. brought up) (no units). The default
        value is vendor/device specific. The penalty value for a
        link up->down state change is nominally 1000 units.";
    }

    leaf suppress {
```

```
    type uint32;
    description
        "Limit at which an interface is suppressed (i.e. held down)
        when its penalty exceeds that limit (no units). The value
        must be greater than the reuse threshold. The default
        value is vendor/device specific. The penalty value for a
        link up->down state change is nominally 1000 units."
    }

    leaf max-suppress-time {
        type uint32;
        units seconds;
        description
            "Maximum time (in seconds) that an interface can be
            suppressed. This value effectively acts as a ceiling that
            the penalty value cannot exceed. The default value is
            vendor/device specific."
    }
}

/*
 * Various types of interfaces support a configurable layer 2
 * encapsulation, any that are supported by YANG should be
 * listed here.
 *
 * Different encapsulations can hook into the common encaps-type
 * choice statement.
 */
augment "/if:interfaces/if:interface" {
    when "if:type = 'ianaift:ethernetCsmacd' or
        if:type = 'ianaift:ieee8023adLag' or
        if:type = 'ethSubInterface' or
        if:type = 'ianaift:pos' or
        if:type = 'ianaift:atmSubInterface'" {
        description "All interface types that can have a configurable
            L2 encapsulation";
        /*
         * TODO - Should we introduce an abstract type to make this
         *         extensible to new interface types, or vendor specific
         *         interface types?
         */
    }

    description "Add encapsulation top level node to interface types
        that support a configurable L2 encapsulation";

    container encapsulation {
```



```
        description
            "Holds the L2 encapsulation associated with an interface";
        choice encaps-type {
            description "Extensible choice of L2 encapsulations";
        }
    }
}

/*
 * Various types of interfaces support loopback configuration, any
 * that are supported by YANG should be listed here.
 */
augment "/if:interfaces/if:interface" {
    when "if:type = 'ianaift:ethernetCsmacd' or
        if:type = 'ianaift:sonet' or
        if:type = 'ianaift:atm' or
        if:type = 'ianaift:otnOtu'" {
        description
            "All interface types that support loopback configuration.";
    }
    if-feature "loopback";
    description "Augments the IETF interface model with loopback
        configuration for interfaces that support it.";

    leaf loopback {
        type identityref {
            base loopback;
        }
        description "Enables traffic loopback.";
    }
}

/*
 * Many types of interfaces support a configurable layer 2 MTU.
 */
augment "/if:interfaces/if:interface" {
    description "Add configurable layer 2 MTU to all appropriate
        interface types.";

    leaf l2-mtu {
        if-feature "configurable-l2-mtu";
        type uint16 {
            range "64 .. 65535";
        }
        description
            "The maximum size of layer 2 frames that may be transmitted
            or received on the interface (excluding any FCS overhead).
            In the case of Ethernet interfaces it also excludes the
```

```
        4-8 byte overhead of any known (i.e. explicitly matched by
        a child sub-interface) 801.1Q VLAN tags.";
    }
}

/*
 * Add generic support for sub-interfaces.
 *
 * This should be extended to cover all interface types that are
 * child interfaces of other interfaces.
 */
augment "/if:interfaces/if:interface" {
    when "derived-from(if:type,
                        'ietf-if-cmn',
                        'sub-interface') or
         if:type = 'ianaift:atmSubInterface' or
         if:type = 'ianaift:frameRelay'" {
        description
            "Any ianaift:types that explicitly represent sub-interfaces
            or any types that derive from the sub-interface identity";
    }
    if-feature "sub-interfaces";
    description "Add a parent interface field to interfaces that
                 model sub-interfaces";
    leaf parent-interface {
        type if:interface-ref;

        mandatory true;
        description
            "This is the reference to the parent interface of this
            sub-interface.";
    }
}

/*
 * Augments the IETF interfaces model with a leaf that indicates
 * which layer traffic is to be transported at.
 */
augment "/if:interfaces/if:interface" {
    if-feature "transport-layer";
    description "Add a top level node to appropriate interfaces to
                 indicate which transport layer an interface is
                 operating at";

    leaf transport-layer {
        type enumeration {
            enum layer-1 {
                value 1;
            }
        }
    }
}
```

```

        description "Layer 1 transport.";
    }
    enum layer-2 {
        value 2;
        description "Layer 2 transport";
    }
    enum layer-3 {
        value 3;
        description "Layer 3 transport";
    }
}
default layer-3;
description
    "The transport layer at which the interface is operating at";
}
}
}
<CODE ENDS>

```

6. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223] for Etherlike interfaces. This includes Ethernet interfaces, 802.3 LAG (802.1AX) interfaces, VLAN sub-interfaces, Switch Virtual interfaces, and Pseudo-Wire Head-End interfaces.

```

<CODE BEGINS> file "interfaces-ethernet-like@2015-06-26.yang"
module interfaces-ethernet-like {
    namespace "urn:ietf:params:xml:ns:yang:interfaces-ethernet-like";
    prefix ethlike;

    import ietf-interfaces {
        prefix if;
    }

    import ietf-yang-types {
        prefix yang;
    }

    import iana-if-type {
        prefix ianaift;
    }

    organization
        "Cisco Systems, Inc.
        Customer Service

        Postal: 170 W Tasman Drive

```

San Jose, CA 95134

Tel: +1 1800 553-NETS

E-mail: cs-yang@cisco.com";

contact

"Robert Wilton - rwilton@cisco.com";

description

"This module contains YANG definitions for configuration for 'Ethernet-like' interfaces. It is applicable to all interface types that use Ethernet framing and expose an Ethernet MAC layer, and includes such interfaces as physical Ethernet interfaces, Ethernet LAG interfaces and VLAN sub-interfaces.";

revision 2015-06-26 {

description "Updated reference to new internet draft name.";

reference

"Internet draft: draft-wilton-netmod-intf-ext-yang-00";

}

/*

* Configuration parameters for Etherlike interfaces.

*/

augment "/if:interfaces/if:interface" {

when "if:type = 'ianaift:ethernetCsmacd' or

if:type = 'ianaift:ieee8023adLag' or

if:type = 'ianaift:l2vlan' or

if:type = 'ianaift:ifPwType'" {

description "Applies to all Ethernet-like interfaces";

}

description

"Augment the interface model with configuration parameters for all Ethernet-like interfaces";

container ethernet-like {

description "Contains configuration parameters for interfaces that use Ethernet framing and expose an Ethernet MAC layer.";

leaf mac-address {

type yang:mac-address;

description

"The configured MAC address of the interface.";

}

}

}

```
/*
 * Operational state for Etherlike interfaces.
 */
augment "/if:interfaces-state/if:interface" {
  when "if:type = 'ianaift:ethernetCsmacd' or
        if:type = 'ianaift:ieee8023adLag' or
        if:type = 'ianaift:l2vlan' or
        if:type = 'ianaift:ifPwType'" {
    description "Applies to all Ethernet-like interfaces";
  }
  description
    "Augments the interface model with operational state parameters
    for all Ethernet-like interfaces.";
  container ethernet-like {
    description "Contains operational state parameters for
    interfaces that use Ethernet framing and expose an
    Ethernet MAC layer.";
    leaf mac-address {
      type yang:mac-address;
      description
        "The operational MAC address of the interface, if
        applicable";
    }

    leaf bia-mac-address {
      type yang:mac-address;
      description
        "The 'burnt-in' MAC address. I.e the default MAC address
        assigned to the interface if none is explicitly
        configured.";
    }
  }
}
}
}
<CODE ENDS>
```

7. Acknowledgements

The authors wish to thank Juergen Schoenwaelder, Mahesh Jethanandani, Michael Zitao, Neil Ketley and Qin Wu for their helpful comments contributing to this draft.

8. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

9. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

9.1. interfaces-common.yang

The interfaces-common YANG module contains various configuration leaves that affect the behavior of interfaces. Modifying these leaves can cause an interface to go down, or become unreliable, or to drop traffic forwarded over it. More specific details of the possible failure modes are given below.

The following leaf could cause the interface to go down, and stop processing any ingress or egress traffic on the interface:

- o /if:interfaces/if:interface/loopback

The following leaf could cause changes to the routing metrics. Any change in routing metrics could cause too much traffic to be routed through the interface, or through other interfaces in the network, potentially causing traffic loss due to excessive traffic on a particular interface or network device:

- o /if:interfaces/if:interface/bandwidth

The following leaves could cause instabilities at the interface link layer, and cause unwanted higher layer routing path changes if the leaves are modified, although they would generally only affect a device that had some underlying link stability issues:

- o /if:interfaces/if:interface/carrier-delay/down

- o /if:interfaces/if:interface/carrier-delay/up

- o /if:interfaces/if:interface/dampening/half-life
- o /if:interfaces/if:interface/dampening/reuse
- o /if:interfaces/if:interface/dampening/suppress
- o /if:interfaces/if:interface/dampening/max-suppress-time

The following leaves could cause traffic loss on the interface because the received or transmitted frames do not comply with the frame matching criteria on the interface and hence would be dropped:

- o /if:interfaces/if:interface/encapsulation
- o /if:interfaces/if:interface/l2-mtu
- o /if:interfaces/if:interface/l3-mtu
- o /if:interfaces/if:interface/transport-layer

Normally devices will not allow the parent-interface leaf to be changed after the interface has been created. If an implementation did allow the parent-interface leaf to be changed then it could cause all traffic on the affected interface to be dropped. The affected leaf is:

- o /if:interfaces/if:interface/parent-interface

9.2. interfaces-ethernet-like.yang

Generally, the configuration nodes in the interfaces-ethernet-like YANG module are concerned with configuration that is common across all types of Ethernet-like interfaces. Currently, the module only contains a node for configuring the operational MAC address to use on an interface. Adding/modifying/deleting this leaf has the potential risk of causing protocol instability, excessive protocol traffic, and general traffic loss, particularly if the configuration change caused a duplicate MAC address to be present on the local network. The following leaf is affected:

- o interfaces/interface/ethernet-like/mac-address

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.

10.2. Informative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Juniper Networks

Email: tsingh@juniper.net

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

R. Wilton, Ed.
D. Ball
Cisco Systems
T. Singh
S. Sivaraj
Juniper Networks
October 19, 2015

Interface VLAN YANG Data Models
draft-wilton-netmod-intf-vlan-yang-01

Abstract

This document defines YANG models to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. Primarily the classification is based on VLAN identifiers in the 802.1Q VLAN tags, but the model also has support for matching on some other layer 2 frame header fields and is designed to be easily extensible to match on other arbitrary header fields.

The model differs from an IEEE 802.1Q bridge model in that the configuration is interface/sub-interface based as opposed to being based on membership of a 802.1Q VLAN bridge.

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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. Terminology	3
1.2. Tree Diagrams	3
2. Objectives	4
3. L3 Interface VLAN Model	4
4. Flexible Encapsulation Model	5
5. L3 Interface VLAN YANG Module	7
6. Flexible Encapsulation YANG Module	9
7. 802.1Q Types YANG Module	17
8. Acknowledgements	22
9. IANA Considerations	22
10. Security Considerations	22
10.1. if-l3-vlan.yang	22
10.2. flexible-encapsulation.yang	23
11. References	24
11.1. Normative References	24
11.2. Informative References	25
Appendix A. Comparison with the IEEE 802.1Q Configuration Model	25
A.1. Sub-interface based configuration model overview	25
A.2. IEEE 802.1Q Bridge Configuration Model Overview	26
A.3. Possible Overlap Between the Two Models	27
Authors' Addresses	27

1. Introduction

This document defines two YANG RFC 6020 [RFC6020] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.wilton-netmod-intf-ext-yang] and the generic interfaces data model defined in RFC 7223 [RFC7223]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature

and service configuration applied to them. In the case of layer 2 Ethernet services, the flexible encapsulation module also supports flexible rewriting of the VLAN tags contained the in frame header.

For reference, a comparison between the sub-interface based YANG model documented in this draft and an IEEE 802.1Q bridge model is described in Appendix A.

In summary, the YANG modules defined in this internet draft are:

if-l3-vlan.yang - Defines the model for basic classification of VLAN tagged traffic to L3 transport services

flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic to L2 transport services

In addition, the yang module dot1q-tag-types.yang, that provides definitions for common ways of identifying frames using fields from the 802.1Q VLAN tag, is included in this draft as a temporary reference. This module may be standardized through IEEE 802.1 and subsequently referenced from this draft rather than included in it.

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 RFC 2119 [RFC2119].

Sub-interface: A sub-interface an a small augmentation of a regular interface in the standard YANG module for Interface Management that represents a subset of the traffic handled by its parent interface. As such, it supports both configuration and operational data using any other YANG models that augment or reference interfaces in the normal way. It is defined in Interface Extensions YANG [I-D.wilton-netmod-intf-ext-yang].

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" means state data (read-only).

- o Symbols after data node names: "?" means an optional node, "!" means a presence container, 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. Objectives

The aim of the YANG models contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices.

The secondary aim is to make the model cleanly extensible, both to handle greater depths of VLAN tag stacks if required, and also to allow vendors to extend the model to include additional forms of tag matching and rewriting if desired.

However, the intention is that it should not be necessary to have any vendor specific extensions to any of the YANG models defined in this document to implement standard Ethernet and VLAN services.

3. L3 Interface VLAN Model

The L3 Interface VLAN model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface based L3 service. It allows for termination of traffic with up to two 802.1Q VLAN tags.

The "if-l3-vlan" YANG module has the following structure:

```
augment /if:interfaces/if:interface/if-cmn:encapsulation/
  if-cmn:encaps-type:
    +--:(vlan)
      +--rw vlan
        +--rw tags
          +--rw tag* [index]
            +--rw index          uint8
            +--rw dot1q-tag
              +--rw tag-type     dot1q-tag-type
              +--rw vlan-id      dot1q-vlan-id
```

4. Flexible Encapsulation Model

The Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 headers before the frame is handed off to the appropriate forwarding code for further handling.

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that manipulations are generally implemented in an asymmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The structure of the model is currently limited to matching or rewriting a maximum of two 802.1Q tags in the frame header but has been designed to be easily extensible to matching/rewriting three or more VLAN tags in future, if required.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

The "flexible-encapsulation" YANG module has the following structure:

```
augment /if:interfaces/if:interface/if-cmn:encapsulation/
  if-cmn:encaps-type:
    +--:(flexible) {flexible-encapsulation-rewrites}?
      +--rw flexible
        +--rw match
          |   +--rw (match-type)
          |   |   +--:(default)
          |   |   |   +--rw default?
          |   |   |   empty
```

```

|      +---:(untagged)
|      | +---rw untagged?          empty
+---:(priority-tagged)
|      +---rw priority-tagged
|      | +---rw tag-type?        dot1q:dot1q-tag-type
+---:(vlan-tagged)
|      +---rw vlan-tagged
|      | +---rw tag* [index]
|      | | +---rw index          uint8
|      | | +---rw dot1q-tag
|      | | | +---rw tag-type      dot1q-tag-type
|      | | | +---rw vlan-id      union
|      | | +---rw match-exact-tags? empty
+---rw rewrite {flexible-rewrites}?
+---rw (direction)?
+---:(symmetrical)
|      +---rw symmetrical
|      | +---rw tag-rewrite {tag-rewrites}?
|      | | +---rw pop-tags?      uint8
|      | | +---rw push-tags* [index]
|      | | | +---rw index        uint8
|      | | | +---rw dot1q-tag
|      | | | | +---rw tag-type    dot1q-tag-type
|      | | | | +---rw vlan-id    dot1q-vlan-id
+---:(asymmetrical) {asymmetric-rewrites}?
+---rw ingress
|      +---rw tag-rewrite {tag-rewrites}?
|      | +---rw pop-tags?      uint8
|      | +---rw push-tags* [index]
|      | | +---rw index        uint8
|      | | +---rw dot1q-tag
|      | | | +---rw tag-type    dot1q-tag-type
|      | | | +---rw vlan-id    dot1q-vlan-id
+---rw egress
|      +---rw tag-rewrite {tag-rewrites}?
|      | +---rw pop-tags?      uint8
|      | +---rw push-tags* [index]
|      | | +---rw index        uint8
|      | | +---rw dot1q-tag
|      | | | +---rw tag-type    dot1q-tag-type
|      | | | +---rw vlan-id    dot1q-vlan-id
augment /if:interfaces/if:interface:
+---rw flexible-encapsulation
+---rw local-traffic-default-encaps
+---rw tag* [index]
|      +---rw index          uint8
+---rw dot1q-tag
|      +---rw tag-type      dot1q-tag-type

```

+-rw vlan-id dot1q-vlan-id

5. L3 Interface VLAN YANG Module

This YANG module augments the encapsulation container defined in Interface Extensions YANG [I-D.wilton-netmod-intf-ext-yang].

```
<CODE BEGINS> file "if-l3-vlan@2015-10-19.yang"
module if-l3-vlan {
  namespace "urn:ietf:params:xml:ns:yang:if-l3-vlan";
  prefix if-l3-vlan;

  import ietf-interfaces {
    prefix if;
  }

  import iana-if-type {
    prefix ianaift;
  }

  import dot1q-tag-types {
    prefix dot1q-tag;
  }

  import interfaces-common {
    prefix if-cmn;
  }

  organization
    "Cisco Systems, Inc.
     Customer Service

     Postal: 170 W Tasman Drive
     San Jose, CA 95134

     Tel: +1 1800 553-NETS

     E-mail: cs-yang@cisco.com";

  contact
    "Robert Wilton - rwilton@cisco.com";

  description
    "This YANG module models L3 VLAN sub-interfaces
    ";

  revision 2015-10-19 {
    description "Latest revision";
```



```
    reference "Internet-Draft draft-wilton-netmod-intf-vlan-yang-01";
  }

  feature l3-vlan-sub-interfaces {
    description
      "This feature indicates that the device supports L3 VLAN
      sub-interfaces";
  }

  /*
   * Add support for the 802.1Q VLAN encapsulation syntax on layer 3
   * terminated VLAN sub-interfaces.
   */
  augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
    "if-cmn:encaps-type" {
    when "../if:type = 'ianaift:l2vlan' and
      ../if-cmn:transport-layer = 'layer-3'" {
      description "Applies only to VLAN sub-interfaces that are
        operating at layer 3";
    }
  }
  if-feature l3-vlan-sub-interfaces;
  description "Augment the generic interface encapsulation with an
    encapsulation for layer 3 VLAN sub-interfaces";

  /*
   * Matches a VLAN, or pair of VLAN Ids to classify traffic
   * into an L3 service.
   */
  case vlan {
    container vlan {
      description
        "Match VLAN tagged frames with specific VLAN Ids";
      container tags {
        description "Matches frames tagged with specific VLAN Ids";
        list tag {
          key "index";
          min-elements 1;
          max-elements 2;
          description "The tags to match, with the outermost tag to
            match with index 0";
          leaf index {
            type uint8 {
              range "0..1";
            }
          }
        }

        /*
         * Only allow matching on an inner tag (at index 1), if
         * also matching on the outer tag at the same time.
         */
      }
    }
  }
}
```

```

        */
    must "index = 0 or
        count ../../tag[index = 0]/index) > 0" {
        error-message
            "An inner tag can only be matched on when also
            matching on an outer tag";
        description
            "Only allow matching on an inner tag, if also
            matching on the outer tag at the same time";
    }
    description
        "The index into the tag stack, outermost tag first";
}

uses dotlq-tag:dotlq-tag;
}
}
}
}
}
}
<CODE ENDS>
```

6. Flexible Encapsulation YANG Module

This YANG module augments the encapsulation container defined in Interface Extensions YANG [I-D.wilton-netmod-intf-ext-yang].

This YANG module also augments the interface container defined in RFC 7223 [RFC7223].

```
<CODE BEGINS> file "flexible-encapsulation@2015-10-19.yang"
module flexible-encapsulation {
  namespace "urn:ietf:params:xml:ns:yang:flexible-encapsulation";
  prefix flex;

  import ietf-interfaces {
    prefix if;
  }

  import interfaces-common {
    prefix if-cmn;
  }

  import dot1q-tag-types {
    prefix dot1q-tag;
  }
}
```

```
organization
  "Cisco Systems, Inc.
    Customer Service

    Postal: 170 W Tasman Drive
    San Jose, CA 95134

    Tel: +1 1800 553-NETS

    E-mail: cs-yang@cisco.com";

contact
  "Robert Wilton - rwilton@cisco.com";

description
  "This YANG module describes interface configuration for flexible
    VLAN matches and rewrites.";

revision 2015-10-19 {
  description "Updated reference to new draft name.";

  reference
    "Internet-Draft draft-wilton-netmod-intf-vlan-yang-01";
}

feature flexible-encapsulation-rewrites {
  description
    "This feature indicates whether the network element supports
      flexible Ethernet encapsulation that allows for matching VLAN
      ranges and performing independent tag manipulations";
}

feature flexible-rewrites {
  description
    "This feature indicates whether the network element supports
      specifying flexible rewrite operations";
}

feature asymmetric-rewrites {
  description
    "This feature indicates whether the network element supports
      specifying different rewrite operations for the ingress
      rewrite operation and egress rewrite operation.";
}

feature tag-rewrites {
  description
    "This feature indicates whether the network element supports
```

```
        the flexible rewrite functionality specifying flexible tag
        rewrites";
    }

    /*
    * flexible-match grouping.
    *
    * This grouping represents a flexible match.
    *
    * The rules for a flexible match are:
    *   1. default, untagged, priority tag, or a stack of tags.
    *   - Each tag in the stack of tags matches:
    *     1. tag type (802.1Q or 802.1ad) +
    *     2. tag value:
    *       i. single tag
    *       ii. set of tag ranges/values.
    *       iii. "any" keyword
    */
    grouping flexible-match {
        description "Flexible match";
        choice match-type {
            mandatory true;
            description "Provides a choice of how the frames may be
                matched";

            case default {
                description "Default match";
                leaf default {
                    type empty;
                    description
                        "Default match. Matches all traffic not matched to any
                        other peer sub-interface by a more specific
                        encapsulation.";
                } // leaf default
            } // case default

            case untagged {
                description "Match untagged Ethernet frames only";
                leaf untagged {
                    type empty;
                    description
                        "Untagged match. Matches all untagged traffic.";
                } // leaf untagged
            } // case untagged

            case priority-tagged {
                description "Match priority tagged Ethernet frames only";
            }
        }
    }
}
```

```
    container priority-tagged {
      description "Priority tag match";
      leaf tag-type {
        type dot1q-tag:dot1q-tag-type;
        description "The 802.1Q tag type of matched priority
                     tagged packets";
      }
    }
  }
}

case vlan-tagged {
  container vlan-tagged {
    description "Matches VLAN tagged frames";
    list tag {
      key "index";
      min-elements 1;
      max-elements 2;
      description "The tags to match, with the outermost tag to
                   match assigned index 0";
      leaf index {
        type uint8 {
          range "0..1";
        }

        must "index = 0 or
              count(..../tag[index = 0]/index) > 0" {
          error-message "An inner tag can only be matched on
                        when also matching on an outer tag";
          description "Only allow matching on an inner tag, if
                       also matching on the outer tags at the
                       same time";
        }
      }
      description
        "The index into the tag stack, outermost tag first";
    }

    uses dot1q-tag:dot1q-tag-ranges-or-any;
  }

  leaf match-exact-tags {
    type empty;
    description
      "If set, indicates that all 802.1Q VLAN tags in the
       Ethernet frame header must be explicitly matched, i.e.
       the EtherType following the matched tags must not be a
       802.1Q tag EtherType.  If unset then extra 802.1Q VLAN
       tags are allowed.";
  }
}
```

```
    }
  }
} // encaps-type
}

/*
 * Grouping for tag-rewrite that can be expressed either
 * symmetrically, or in the ingress and/or egress directions
 * independently.
 */
grouping tag-rewrite {
  description "Flexible rewrite";
  leaf pop-tags {
    type uint8 {
      range 1..2;
    }
    description "The number of tags to pop (or translate if used in
      conjunction with push-tags)";
  }

  list push-tags {
    key "index";
    max-elements 2;
    description "The number of tags to push (or translate if used
      in conjunction with pop-tags)";
  }

  /*
   * Server should order by increasing index.
   */
  leaf index {
    type uint8 {
      range 0..1;
    }

    /*
     * Only allow a push of an inner tag if an outer tag is also
     * being pushed.
     */
    must "index != 0 or
      count ../../push-tags[index = 0]/index > 0" {
      error-message "An inner tag can only be pushed if an outer
        tag is also specified";
      description "Only allow a push of an inner tag if an outer
        tag is also being pushed";
    }
    description "The index into the tag stack";
  }
}

uses dot1q-tag:dot1q-tag;
```

```
    }
  }

  /*
   * Grouping for all flexible rewrites of fields in the L2 header.
   *
   * This currently only includes flexible tag rewrites, but is
   * designed to be extensible to cover rewrites of other fields in
   * the L2 header if required.
   */
  grouping flexible-rewrite {
    description "Flexible rewrite";

    /*
     * Tag rewrite.
     *
     * All tag rewrites are formed using a combination of pop-tags
     * and push-tags operations.
     */
    container tag-rewrite {
      if-feature tag-rewrites;
      description "Tag rewrite. Translate operations are expressed
        as a combination of tag push and pop operations.";
      uses tag-rewrite;
    }
  }

  augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
    "if-cmn:encaps-type" {
    when "../if:if-cmn = 'if-cmn:ethSubInterface' and
      ../if-cmn:transport-layer = 'layer-2'" {
      description "Applies only to Ethernet sub-interfaces that are
        operating at transport layer 2";
    }
    description
      "Add flexible match and rewrite for VLAN sub-interfaces";

    /*
     * A flexible encapsulation allows for the matching of ranges and
     * sets of VLAN Ids. The structure is also designed to be
     * extended to allow for matching/rewriting other fields within
     * the L2 frame header if required.
     */
    case flexible {
      if-feature flexible-encapsulation-rewrites;
      description "Flexible encapsulation and rewrite";
      container flexible {
        description "Flexible encapsulation and rewrite";
      }
    }
  }
}
```

```

    container match {
        description
            "The match used to classify frames to this interface";
        uses flexible-match;
    }

    container rewrite {
        if-feature flexible-rewrites;
        description "L2 frame rewrite operations";
        choice direction {
            description "Whether the rewrite policy is symmetrical or
                asymmetrical";
            case symmetrical {
                container symmetrical {
                    uses flexible-rewrite;
                    description
                        "Symmetrical rewrite. Expressed in the ingress
                        direction, but the reverse operation is applied
                        to egress traffic";
                }
            }

            /*
             * Allow asymmetrical rewrites to be specified.
             */
            case asymmetrical {
                if-feature asymmetric-rewrites;
                description "Asymmetrical rewrite";
                container ingress {
                    uses flexible-rewrite;
                    description "Ingress rewrite";
                }
                container egress {
                    uses flexible-rewrite;
                    description "Egress rewrite";
                }
            }
        }
    }
}

augment "/if:interfaces/if:interface" {
    when "if:type = 'if-cmn:ethSubInterface' and
        if-cmn:transport-layer = 'layer-2'" {
        description "Any L2 Ethernet sub-interfaces";
    }
}

```



```
description "Add flexible encapsulation configuration for VLAN
            sub-interfaces";

/*
 * All flexible encapsulation specific interface configuration
 * (except for the actual encapsulation and rewrite) is contained
 * by a flexible-encapsulation container on the interface.
 */
container flexible-encapsulation {
  description
    "All per interface flexible encapsulation related fields";

  /*
   * For encapsulations that match a range of VLANs (or Any),
   * allow configuration to specify the default VLAN tag values
   * to use for any traffic that is locally sourced from an
   * interface on the device.
   */
  container local-traffic-default-encaps {
    description "The VLAN tags to use by default for locally
                sourced traffic";

    list tag {
      key "index";
      max-elements 2;

      description
        "The VLAN tags to use by locally sourced traffic";

      leaf index {
        type uint8 {
          range "0..1";
        }

        /*
         * Only allow an inner tag to be specified if an outer
         * tag has also been specified.
         */
        must "index = 0 or
              count(..../tag[index = 0]/index) > 0" {
          error-message "An inner tag can only be specified if an
                        outer tag has also been specified";
          description "Ensure that an inner tag cannot be
                      specified without an outer tag";
        }

        description "The index into the tag stack, outermost tag
                    assigned index 0";
      }
    }
  }
}
```

```

    uses dot1q-tag:dot1q-tag;
  }
}
}
}
}
<CODE ENDS>

```

7. 802.1Q Types YANG Module

This YANG module has no external imports.

The expectation is that the raw 802.1Q VLAN tag fields types will eventually be standardized in IEEE rather than IETF. They are included here to make the model complete.

The groupings that can be used to generally identify frames based on the fields in the 802.1Q tag may be defined here or in IEEE depending on where is deemed appropriate after discussion with IEEE 802.1.

```
<CODE BEGINS> file "dot1q-tag-types@2015-10-19.yang"
module dot1q-tag-types {
  namespace "urn:ieee:params:xml:ns:yang:dot1q-tag-types";
  prefix dot1q;

  import ieee-types {
    prefix ieee-types;
  }

  import ieee-dot1q-types {
    prefix dot1q-types;
  }

  organization
    "Cisco Systems, Inc.
     Customer Service

     Postal: 170 W Tasman Drive
     San Jose, CA 95134

     Tel: +1 1800 553-NETS

     E-mail: cs-yang@cisco.com";

  contact
    "Robert Wilton - rwilton@cisco.com";

  description
```

"This module contains a collection of generally useful YANG types that are specific to 802.1Q VLANs that can be usefully shared between multiple models.

Terms and Acronyms

802.1Q: IEEE 802.1Q VLANs

VLAN (vlan): Virtual Local Area Network
";

```
revision 2015-10-19 {
  description "Latest revision, references IEEE types";

  reference "Currently defined in:
    Internet-Draft draft-wilton-netmod-intf-vlan-yang-01,
    but may be standardized in IEEE 802.1";
}

typedef PCP {
  type uint8 {
    range "0..7";
  }
  description
    "Priority Code Point. PCP is a 3-bit field that refers to the
    class of service applied to an 802.1Q VLAN tagged frame. The
    field specifies a priority value between 0 and 7, these values
    can be used by quality of service (QoS) to prioritize
    different classes of traffic.";
  reference "IEEE 802.1Q (2014)";
}

/*
 * Defines the supported IEEE 802.1Q types that can be used for
 * VLAN tag matching.
 */
identity dot1q-tag-vlan-type {
  description "Base identity from which all 802.1Q VLAN tag types
    are derived from";
}

identity c-vlan {
  base dot1q-tag-vlan-type;
  description
    "An 802.1Q Customer-VLAN tag, normally using the 0x8100
    Ethertype";
}
```

```
identity s-vlan {
  base dot1q-tag-vlan-type;
  description
    "An 802.1Q Service-VLAN tag, using the 0x88a8 Ethertype
    originally introduced in 802.1ad, and incorporated into
    802.1Q (2011)";
}

typedef dot1q-tag-type {
  type identityref {
    base "dot1q-tag-vlan-type";
  }
  description "Identifies a specific 802.1Q tag type";
  reference "IEEE 802.1Q (2014)";
}

/*
 * A grouping which represents an 802.1Q VLAN tag, matching both
 * the tag Ethertype and a single VLAN Id. The PCP and DEI fields
 * in the 802.1Q tag are ignored for tag matching purposes.
 */
grouping dot1q-tag {
  description "Grouping to allow configuration to identify a single
    802.1Q VLAN tag";
  container dot1q-tag {
    description "Identifies an 802.1Q VLAN tag with an explicit
      tag-type and a single VLAN Id";
    leaf tag-type {
      type dot1q-tag-type;
      mandatory true;
      description "VLAN tag type";
    }
    leaf vlan-id {
      type ieee-types:vlanid;
      mandatory true;
      description "VLAN Id";
    }
  }
}

/*
 * A grouping which represents an 802.1Q VLAN tag, matching both
 * the tag Ethertype and a single VLAN Id or "any" to match on any
 * VLAN Id. The PCP and DEI fields in the 802.1Q tag are ignored
 * for tag matching purposes.
 */
grouping dot1q-tag-or-any {
```

```

    description "Grouping to allow configuration to identify a single
        802.1Q VLAN tag or the 'any' value to match any VLAN
        Id not matched by a more specific VLAN Id match";
    container dot1q-tag {
        description "Identifies an 802.1Q VLAN tag with an explicit
            tag-type and a single VLAN Id, or 'any' VLAN Id";
        leaf tag-type {
            type dot1q-tag-type;
            mandatory true;
            description "VLAN tag type";
        }
        leaf vlan-id {
            type union {
                type ieee-types:vlanid;
                type enumeration {
                    enum "any" {
                        value 4096;
                        description
                            "Matches 'any' VLAN tag in the range 1 to 4094 that
                             is not matched by a more specific VLAN Id match";
                    }
                }
            }
            mandatory true;
            description "VLAN Id or any";
        }
    }
}

/*
 * A grouping which represents an 802.1Q tag that matches a range
 * of VLAN Ids. The PCP and DEI fields in the 802.1Q tag are
 * ignored for tag matching purposes.
 */
grouping dot1q-tag-ranges {
    description "Grouping to allow configuration to identify an
        802.1Q VLAN tag that matches any VLAN Id within a
        set of non overlapping VLAN Id ranges";
    container dot1q-tag {
        description "Identifies an 802.1Q VLAN tag with an explicit
            tag-type and and a range of VLAN Ids";
        leaf tag-type {
            type dot1q-tag-type;
            mandatory true;
            description "VLAN tag type";
        }
        leaf vlan-ids {
            type dot1q-types:vid-range;
        }
    }
}

```

```

        mandatory true;
        description "VLAN Ids";
    }
}

/*
 * A grouping which represents an 802.1Q VLAN tag, matching both
 * the tag Ethertype and a single VLAN Id, ordered list of ranges,
 * or "any" to match on any VLAN Id. The PCP and DEI fields in the
 * 802.1Q tag are ignored for tag matching purposes.
 */
grouping dot1q-tag-ranges-or-any {
    description "Grouping to allow configuration to identify an
        802.1Q VLAN tag that matches any specific VLAN Id
        within a set of non overlapping VLAN Id ranges, or
        the 'any' value to match any VLAN Id";
    container dot1q-tag {
        description "Identifies an 802.1Q VLAN tag with an explicit
            tag-type, an ordered list of VLAN Id ranges, or
            'any' VLAN Id";
        leaf tag-type {
            type dot1q-tag-type;
            mandatory true;
            description "VLAN tag type";
        }
        leaf vlan-id {
            type union {
                type dot1q-types:vid-range;
                type enumeration {
                    enum "any" {
                        description "Matches 'any' VLAN tag in the range 1 to
                            4094";
                    }
                }
            }
            mandatory true;
            description "VLAN Ids or any";
        }
    }
}
}
<CODE ENDS>

```

8. Acknowledgements

The authors Neil Ketley for his helpful comments contributing to this draft.

9. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

10. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

10.1. if-l3-vlan.yang

The nodes in the if-l3-vlan YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/vlan, that are sensitive to this are:

- o tags
- o tags/index
- o tags/index/tag-type
- o tags/index/vlan-id

10.2. flexible-encapsulation.yang

There are many nodes in the flexible-encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree `/interfaces/interface/encapsulation/flexible/match`, that are sensitive to this are:

- o `default`
- o `untagged`
- o `priority-tagged`
- o `priority-tagged/tag-type`
- o `vlan-tagged`
- o `vlan-tagged/index`
- o `vlan-tagged/index/dot1q-tag/vlan-type`
- o `vlan-tagged/index/dot1q-tag/vlan-id`
- o `vlan-tagged/match-exact-tags`

There are also many nodes in the flexible-encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree `/interfaces/interface/encapsulation/flexible/rewrite`, that are sensitive to this are:

- o `symmetrical/tag-rewrite/pop-tags`
- o `symmetrical/tag-rewrite/push-tags`
- o `symmetrical/tag-rewrite/push-tags/index`
- o `symmetrical/tag-rewrite/push-tags/dot1q-tag/tag-type`
- o `symmetrical/tag-rewrite/push-tags/dot1q-tag/vlan-id`

- o asymmetrical/ingress/tag-rewrite/pop-tags
- o asymmetrical/ingress/tag-rewrite/push-tags
- o asymmetrical/ingress/tag-rewrite/push-tags/index
- o asymmetrical/ingress/tag-rewrite/push-tags/dot1q-tag/tag-type
- o asymmetrical/ingress/tag-rewrite/push-tags/dot1q-tag/vlan-id
- o asymmetrical/egress/tag-rewrite/pop-tags
- o asymmetrical/egress/tag-rewrite/push-tags
- o asymmetrical/egress/tag-rewrite/push-tags/index
- o asymmetrical/egress/tag-rewrite/push-tags/dot1q-tag/tag-type
- o asymmetrical/egress/tag-rewrite/push-tags/dot1q-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

- o tag
- o tag/index
- o tag/dot1q-tag/tag-type
- o tag/dot1q-tag/vlan-id

11. References

11.1. Normative References

- [I-D.wilton-netmod-intf-ext-yang]
Wilton, R., Ball, D., Singh, T., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-wilton-netmod-intf-ext-yang-00 (work in progress), July 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.

11.2. Informative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group is also developing a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- o The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.

- o Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.
- o Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.
- o In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.
- o The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- o Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.
- o Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.
- o Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.
- o VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.

- o Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Juniper Networks

Email: tsingh@juniper.net

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net