NETMOD Working Group
INTERNET-DRAFT
Intended Status: Standards Track

Anil Kumar S N
Gaurav Agrawal
Vinod Kumar S
Huawei Technologies

Expires: January 1, 2017

June 30, 2016

YANG compiler annotation for data structure and inheritance
draft-agv-netmod-yang-annotation-ds-and-derived-00

Abstract

This document defines two new YANG compiler annotations as per draft-agv-netmod-yang-compiler-metadata-00. First annotation is used to define the data structure type to be generated corresponding to a schema node. Second annotation is used to generate a user defined inherited class corresponding to a schema node in which user can override the default implementation.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 1, 2017

Table of Contents

1 Introduction

YANG defines single instance data structure as container or leaf and multi instance data structure as a list or leaf-list. Mapping of the YANG single instance constructs to a programming data structure is straight forward, it can be directly mapped to an object or an entity.

Mapping of the YANG multi instance construct to a programming data structure has many option, since it is a collection or objects / entities. Depending on the application use case and development environment, it will have other requirements to be considered for mapping it to a data structure. For example it needs to be optimized for storage or it need to be optimized for search efficiency or it may be required to have multiple key combinations for search. Based on this, we can see that there is no single option to be used by YANG utilities / compilers to auto generate the code corresponding to a multi instance construct.

Applications use the YANG to document the external interface designed. Applications need to design the alternatives for data structure and choose that is best suited for them. This is part of typical software engineering activity used by application. YANG utilities / compilers need to get these design related details to automate the code generation as per application design. To support this a new YANG compiler annotation @app-data-structure is defined.

Applications need to extend the auto generated code to suit their needs, The external world communication using protocol like NETCONF / RESTONF can be automated by automating the code generation based on the YANG structure. Applications have additional business logic to be taken care, wherein they need to extend the generated code. When applications extends or modifies the generated code, YANG utilities are unaware of it, and will not be able to correctly update the auto generated files when the schema changes. In such scenarios, the YANG utilities / compilers generated code needs to be provide a framework wherein application can extend the default auto-generated implementation provided by the utilities which is not impacted even if YANG utilities / compilers auto generates files for changed schema. To support this a new YANG compiler annotation @app-derived is defined.

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 RFC 2119 [RFC2119].

2.2 Terms Defined in Other Documents

The following terms are defined in [RFC6241]:

- capability,
- client,
- datastore,
- message,
- protocol operation,
- server.

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- action,
- anydata,
- anyxml,
- built-in type,
- container,
- data model,
- data node,
- data tree,
- derived type,
- extension,
- leaf,

- leaf-list,
- list,
- module,
- RPC input and output.

2.4 Definitions of New Terms

- @app-data-structure: annotations for type of data structure to be generated for multi instance YANG construct.
- @app-derived: annotation for derived class to be generated for extending the generated class.

3. Defining @app-data-structure compiler annotations in YANG

@app-data-structure annotation is used to define the data structure to be used for the corresponding multi-instance YANG construct.

It has the following parameter.

- data-structure: Its value will specify the data structure to be used for code generation. It is a mandatory parameter.
- key-fields: This is used to specify the space separated list of key fields. This is not applicable for leaf-list YANG construct, it is optional parameter. If it is not defined, the list's key fields will be used to index the data structure.

3.1 Example usage

Application instructs to use a map data structure for maintaining servers information.

```
list server {
   ca:compiler-annotation{
     @app-data-structure(data-structure="map", key="name");
   }

   key "name";
   unique "ip port";
   leaf name {
     type string;
   }
   leaf ip {
```

```
          type inet:ip-address;
        }
        leaf port {
          type inet:port-number;
        }
      }
```

4 Defining @app-derived compiler annotations in YANG

@app-derived is used to generate an inherited class, which can be used by the applications to extend/override the default implementations of application interface. It has the following parameter.

o extended-name: It is used to generate the extended class, which can be used by application for implementation. This is a mandatory parameter.

4.1 Example usage

Application instructs to generate an inherited class for implementation.

```
      list server {
        ca:compiler-annotation{
          @app-data-structure(data-structure="map", key="name");
          @app-derived(extended-name=special-server);
        }

        key "name";
        unique "ip port";
        leaf name {
          type string;
        }
        leaf ip {
          type inet:ip-address;
        }
        leaf port {
          type inet:port-number;
        }
      }
```

5 Security Considerations

This document introduces two annotations for defining compiler metadata in YANG modules and attaching them to instances of YANG schema nodes. By itself, this mechanism represents no security threat.

6 IANA Considerations

No specific IANA considerations for this document

7. Acknowledgments

8 References

8.1 Normative References

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-11 (work in progress), February 2016.

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-09 (work in progress), March 2016.

[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>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

8.2 Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-10 (work in progress), March 2016.

[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>.

Authors' Addresses

Anil Kumar S N
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560037

EMail: anil.ietf@gmail.com

Gaurav Agrawal
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560037

EMail: gaurav.agrawal@huawei.com

Vinod Kumar S
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560037

EMail: vinods.kumar@huawei.com

Internet Draft
NETMOD Working Group
Category: Informational
Expires: January 9, 2017

Vinod Kumar S
Gaurav Agrawal
Anil Kumar S N

July 8, 2016

Defining and Using Metadata for YANG compilers
draft-agv-netmod-yang-compiler-metadata-01

Abstract

This document defines mechanism to defining compiler metadata (annotations) in YANG modules using YANG extension statement.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 9, 2017.

Table of Contents

1. Introduction

YANG started as a data modeling language used to model configuration data, state data, remote procedure calls, and notifications for network management protocols. The purpose of YANG was focused on expressing the data organization while being exchanged over network. It also addressed the constraints that needs to be taken care by the data set exchanged.

YANG has out grown the purpose for which it was invented. Due to its simplicity in structure and flexibility many tools have been developed which tries to ease the application development by automating the code generated corresponding to defined schema.

All the implementation related data structures / classes could be auto generated and applications only concentrate on the business logic implementation. Applications are being relieved from actual protocol implementation for exchanging information with external system.

The purpose of YANG was focused on expressing the data organization while being exchanged over network. Hence the scope of automation in application development cannot cater to data organization / processing within the system.

This gap needs to be addressed in a standardized way so that it’s not compiler / utilities / platform / language specific. This enable application to be portable across multiple platforms without any additional effort with respect to data organization.

Also it is required that the mechanism of annotation should not be in-line with original YANG module/sub-module, so that it will not result in maintenance issues.

So there is a need to support compiler annotations in YANG, by which applications can instruct the YANG utilities or compilers to automate the application development related to data organization / processing. These annotations should be maintained in additional YANG module / sub-module which can be optionally consumed by supporting compilers.

Typical use cases are:

- Feed input to YANG compiler/utilities which can be used to automate the code generation based on standard data structure or collections.

- Enable the code generation to incorporate the design patterns required by applications.

- Enable the data structure or collections to have multiple indexes beyond the current supported list’s key(s). Since the actual implementation would required searching the data based on different leaf combinations.

- Enable applications to model internal data organization, required for business logic implementation, and not exposed to outside world.

Usage of compiler annotations are dependent on the compiler consuming it. This draft is intended to document YANG extension to support

defining compiler annotation framework.It is outside the scope of this document about the specific compiler annotation(s) definition / usage. Individual annotation definition and usage SHOULD be standardized in other docs.

Definition and usage of compiler annotation is limited to a particular protocol or application development within a device, it has no effect on how the management information is exchanged between 2 devices over network. A server SHOULD share its YANG file(s) after removing the compiler annotations that was added for its implementation. A client MUST ignore any compiler annotations present in the YANG file(s). A client MAY redefine the compiler annotation as per its implementation requirements. Clients MAY also add new annotation depending on its implementation requirements.

This document proposes a systematic way for defining compiler metadata annotations. For this purpose, YANG extension statement "compiler-annotation" is defined in the module "agv-yang-compiler-annotation" (Section 5). Other YANG modules importing this module can use the "compiler-annotation" statement for defining one or more compiler annotations.

The benefits of defining the compiler-annotations in a YANG module are the following:

- Applications can use YANG as a tool to design the application implementation.

- Enhance the YANG compiler(s) capability to automate the application development process.

- Enhance the protocol development to provide better application development framework.

- YANG could be extended to support data modeling for protocol beyond NETCONF or RESTCONF.

Due to the rules for YANG extensions (see sec. 6.3.1 in [I-D.ietf-netmod-rfc6020bis]), compiler-annotation definitions posit relatively weak conformance requirements. The alternative of introducing a new built-in YANG mechanism for compiler annotations was considered, but it was seen as a major change to the language that is inappropriate for YANG 1.1, which was chartered as a maintenance revision. After evaluating real-life usage of compiler metadata annotations, it is designed in such way that the ABNF grammar can seamlessly adapt the current defined compiler annotations.

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]:

- capability,
- client,
- datastore,
- message,
- protocol operation,
- server.

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- action,
- anydata,
- anyxml,
- built-in type,
- container,
- data model,
- data node,
- data tree,
- derived type,
- extension,
- leaf,

- leaf-list,
- list,
- module,
- RPC input and output.

2.4. Definitions of New Terms

- compiler-annotation: a single item of compiler metadata that is attached to YANG constructs.
- compiler metadata: additional information that complements a schema tree.

3. Defining compiler annotations in YANG

Compiler metadata annotations are defined by YANG extension statement "ca:compiler-annotation". This YANG language extension is defined in the module "ietf-yang-compiler-annotation" (Section 5).

Sub-statements of "ca:compiler-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.

sub-statement	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
units	7.3.3	0..1
...	Current Section 5	1..n

Table 2: Substatements of "ca:compiler-annotation".

This draft only specifies a mechanism to define compiler metadata (annotations) in YANG modules using YANG extension statement. It provides a generic extension based mechanism, to define the annotations, specific extensions needs to be defined for specific data organization annotations as sub statement to compiler annotation extension.

An compiler annotation can be made conditional by using one or more "if- feature" statements; the compiler annotation is then consumed by

compilers and perform the desired operation in compilation.

The semantics and usage rules for a specific compiler-annotation extensions SHOULD be fully specified in another document.

A compiler-annotation MUST NOT change the schema tree semantics defined by YANG. For example, it is illegal to define and use an compiler-annotation that allows modification to data-def-stmts.

The "status" statement can be used exactly as for YANG schema nodes.

3.1. Example Definition

```
module example-yang {

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

  prefix "example-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>";

  description
       "This YANG module demonstrates the usage of compiler
       annotation by any module.";

  revision 2016-07-08 {
       description
         "Initial revision.";
       reference
         "draft-agv-netmod-yang-compiler-metadata: example YANG which
          is annotated";
  }

  container candidate-servers {
     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 module defines the "app-data-structure" compiler-annotation as specific compiler annotation extension:

```
module example-compiler-annotation {

namespace "urn:ietf:params:xml:ns:yang:example-compiler-annotation";

prefix "example";

import ietf-yang-compiler-annotation {
       prefix "ca";
}

import ietf-yang-app-data-structure-annotation {
       prefix "ds";
}

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

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

     description
       "This YANG module demonstrates the usage of compiler
       annotation by any module.";

     revision 2016-07-08 {
       description
         "Initial revision.";
       reference
         "draft-agv-netmod-yang-compiler-metadata: example of
          defining and using compiler annotations with YANG";
     }

ca:compiler-annotation /candidate-servers/server {
       ds:app-data-structure queue;
     }
}
```

4. Using Annotations

By defining a YANG module in which a compiler metadata annotation is defined using the "ca:compiler-annotation" statement, an application indicates compiler to handle that compiler-annotation according to the compiler-annotation’s definition. That is, the compiler-annotation uses it as input for automation of code generation or applications development.

Depending on its semantics, an annotation may have an effect only in certain schema trees and/or on specific schema node types.

A client MUST NOT use the compiler-annotation to interpret the schema even if it is advertised by a server.

5. Metadata YANG Module

FC Editor: In this section, replace all occurrences of ’XXXX’ with he actual RFC number and all occurrences of the revision date below ith the date of RFC publication (and remove this note).

FC Editor: Also please replace all occurrences of ’RFC 6020bis’ with he actual RFC number that will be assigned to [I-D.ietf-netmod-fc6020bis].

```
CODE BEGINS> file "ietf-yang-compiler-annotation.yang"

odule ietf-yang-compiler-annotation {

amespace "urn:ietf:params:xml:ns:yang:ietf-yang-compiler-annotation";

refix "ca";

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

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

escription
 "This YANG module defines an extension statement that allows for
  defining compiler annotations.

  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).";
```

```
    revision 2016-07-08 {
    description
      "Initial revision.";
    reference
      "draft-agv-netmod-yang-compiler-metadata:
       Defining and Using compiler annotations with YANG";
    }

    extension compiler-annotation {
    argument target;
    description
      "This extension allows for defining compiler annotations for
       any body-stmts. The 'ca:compiler-annotation' statement
       contains annotations applicable to its target statement
       identified by the argument.

       It's purpose is to provide additional information to compiler
       about implementation of the modeled information.

   he argument is a string that identifies a node in the
   chema tree.  This node is called the compiler annotation's
       target node.  The target node MUST be a body-stmt as defined
       in RFC6020bis.

       It MAY be consumed by the compiler of the device supporting
       the schema.

       The compiler annotations must be defined in a seperate YANG file,
       so that there are no maintenance issues.

       The ca:compiler-annotation defined with this extension
       statement do not affect the namespace or get impacted by
       the namespace of the YANG file where it is used.

       Semantics of the annotation and other documentation can be
       specified using the following standard YANG substatements (all
       are optional):  'description', 'reference', 'status', and
       'units'.

       The presence of a 'if-feature' child to the ca:
       compiler-annotation, means the compiler consumes the
       annotation when the feature is supported by the device.

       Server SHOULD NOT share ca:compiler-annotations YANG files
       while sharing schema with a client in protocol exchange.

       Client receiving the schema from a Server in protocol
       exchange, MUST ignore the YANG files with any
```

```
      ca:compiler-annotations extension.

      There must be one or more sub statements with specific compiler
      annotation extensions. (Note: Specific compiler annotation
      extensions SHOULD be covered as a part of other standard
      documents.)

    } // compiler-annotation
  //module agv-yang-compiler-annotation

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:agv:params:xml:ns:yang:agv-yang-compiler-annotation

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:         agv-yang-compiler-annotation
namespace:    urn:agv:params:xml:ns:yang:
                                      agv-yang-compiler-annotation
prefix:       md
reference:    RFC XXXX
---------------------------------------------------------------------
```

9 Security Considerations

This document introduces a mechanism for defining compiler metadata annotations in YANG modules and attaching them to instances of YANG schema nodes. By itself, this mechanism represents no security threat. Security implications of a particular compiler-annotation defined using this mechanism MUST be duly considered and documented in the the compiler-annotation’s definition.

10. Acknowledgments

11 References

11.1 Normative References

I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-11 (work in progress), February 2016.

I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-09 (work in progress), March 2016.

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>.

11.2 Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-10 (work in progress), March 2016.

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

[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>.

[RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, May 1998.

Authors' Addresses

Vinod Kumar S
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560066

EMail: vinods.kumar@huawei.com

Gaurav Agrawal
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560066

EMail: gaurav.agrawal@huawei.com

Anil Kumar S N
Huawei Technologies India Pvt. Ltd,
Near EPIP Industrial Area,
Kundalahalli Village,
Whitefield,
Bangalore - 560066

EMail: anil.ietf@gmail.com

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2017

J. Haas
Juniper
S. Hares
Huawei

July 6, 2016

I2RS Ephemeral State Requirements
draft-ietf-i2rs-ephemeral-state-14

Abstract

This document covers requests to the NETMOD and NETCONF Working Groups for functionality to support the ephemeral state requirements to implement the I2RS architecture.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) Working Group is chartered with providing architecture and mechanisms to inject into and retrieve information from the routing system. The I2RS Architecture document [I-D.ietf-i2rs-architecture] abstractly documents a number of requirements for implementing the I2RS requirements. Section 2 reviews 10 key requirements related to ephemeral state.

The I2RS Working Group has chosen to use the YANG data modeling language [RFC6020] as the basis to implement its mechanisms.

Additionally, the I2RS Working group has chosen to re-use two existing protocols, NETCONF [RFC6241] and its similar but lighter-weight relative RESTCONF [I-D.ietf-netconf-restconf], as the protocols for carrying I2RS.

What does re-use of a protocol mean? Re-use means that while YANG, NETCONF and RESTCONF are a good starting basis for the I2RS protocol, the creation of the I2RS protocol implementations requires that the I2RS requirements

1. select features from YANG, NETCONF, and RESTCONF per version of the I2RS protocol (See sections 4, 5, and 6)

2. propose additions to YANG, NETCONF, and RESTCONF per version of the I2RS protocol for key functions (ephemeral state, protocol security, publication/subscription service, traceability),

3. suggest protocol strawman (e.g. [I-D.hares-i2rs-protocol-strawman]) as ideas for the NETCONF, RESTCONF, and YANG changes.

The purpose of these requirements and the suggested protocol straw man is to provide a quick turnaround on creating the I2RS protocol.

Support for ephemeral state is an I2RS protocol requirement that requires datastore changes (see section 3), YANG additions (see section 4), NETCONF additions (see section 5), and RESTCONF additions (see section 6).

Sections 7-9 provide details that expand upon the changes in sections 3-6 to clarify requirements discussed by the I2RS and NETCONF working groups. Sections 7 provide additional requirements that detail how write-conflicts should be resolved if two I2RS client write the same data. Section 8 provides an additional requirement that details on I2RS support of multiple message transactions. Section 9 highlights two requirements in the I2RS publication/subscription requirements [I-D.ietf-i2rs-pub-sub-requirements] that must be expanded for ephemeral state.

1.1. Requirements Language

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

2. Review of Requirements from I2RS architecture document

The I2RS architecture defines important high-level requirements for the I2RS protocol. The following are ten requirements that [I-D.ietf-i2rs-architecture] contains which provide context for the ephemeral data state requirements given in sections 3-8:

1. The I2RS protocol SHOULD support highly reliable notifications (but not perfectly reliable notifications) from an I2RS agent to an I2RS client.

2. The I2RS protocol SHOULD support a high bandwidth, asynchronous interface, with real-time guarantees on getting data from an I2RS agent by an I2RS client.

3. The I2RS protocol will operate on data models which MAY be protocol independent or protocol dependent.

4. I2RS Agent MUST record the client identity when a node is created or modified. The I2RS Agent SHOULD to be able to read the client identity of a node and use the client identity's associated priority to resolve conflicts. The secondary identity is useful for traceability and may also be recorded.

5. Client identity MUST have only one priority for the client's identifier. A collision on writes is considered an error, but the priority associated with each client identifier is utilized to compare requests from two different clients in order to modify an existing node entry. Only an entry from a client which is higher priority can modify an existing entry (First entry wins). Priority only has meaning at the time of use.

6. The Agent identity and the Client identity SHOULD be passed outside of the I2RS protocol in a authentication and authorization protocol (AAA). Client priority may be passed in the AAA protocol. The values of identities are originally set by operators, and not standardized.

7. An I2RS Client and I2RS Agent MUST mutually authenticate each other based on pre-established authenticated identities.

8. Secondary identity data is read-only meta-data that is recorded by the I2RS agent associated with a data model's node is written, updated or deleted. Just like the primary identity, the secondary identity SHOULD only be recorded when the data node is written or updated or deleted

9. I2RS agent MAY have a lower priority I2RS client attempting to modify a higher priority client's entry in a data model. The filtering out of lower priority clients attempting to write or modify a higher priority client's entry in a data model SHOULD be effectively handled and not put an undue strain on the I2RS agent.

10. The I2RS protocol MUST support the use of a secure transport. However, certain functions such as notifications MAY use a non-secure transport. Each model or service (notification, logging) must define within the model or service the valid uses of a non-secure transport.

3. Ephemeral State Requirements

In requirements Ephemeral-REQ-01 to Ephemeral-15, Ephemeral state is defined as potentially including both ephemeral configured state and operational state.

3.1. Persistence

Ephemeral-REQ-01: I2RS requires ephemeral state; i.e. state that does not persist across reboots. If state must be restored, it should be done solely by replay actions from the I2RS client via the I2RS agent.

While at first glance this may seem equivalent to the writable-running data store in NETCONF, running-config can be copied to a persistent data store, like startup config. I2RS ephemeral state MUST NOT be persisted.

3.2. Constraints

Ephemeral-REQ-02: Non-ephemeral state MUST NOT refer to ephemeral state for constraint purposes; it SHALL be considered a validation error if it does.

Ephemeral-REQ-03: Ephemeral state may have constraints that refer to operational state, this includes potentially fast changing or short lived operational state nodes, such as MPLS LSP-ID or a BGP IN-RIB. Ephemeral state constraints should be assessed when the ephemeral state is written, and if any of the constraints change to make the constraints invalid after that time the I2RS agent should notify the I2RS Client.

Ephemeral-REQ-04: Ephemeral state MUST be able to refer to non-ephemeral state as a constraint. Non-ephemeral state can be configuration state or operational state.

Ephemeral-REQ-05: I2RS pub-sub, logging, RPC or other mechanisms may lead to undesirable or unsustainable resource consumption on a system implementing an I2RS Agent. It is RECOMMENDED that mechanisms be made available to permit prioritization of I2RS operations, when appropriate, to permit implementations to shed work load when operating under constrained resources. An example of such a work shedding mechanism is rate-limiting.

3.3. Hierarchy

Ephemeral-REQ-06: The ability to:

1. to define a YANG module or submodule schema that only contains data nodes with the property of being ephemeral, and

2. to augment a YANG data model with additional YANG schema nodes that have the property of being ephemeral.

3.4. Ephemeral Configuration overlapping Local Configuration

Ephemeral-REQ-07: Ephemeral configuration state could override overlapping local configuration state, or vice-versa. Implementations MUST provide a mechanism to choose which takes precedence. This mechanism MUST include local configuration (policy) and MAY be provided via the I2RS protocol mechanisms.

4. YANG Features for Ephemeral State

Ephemeral-REQ-08:In addition to config true/false, there MUST be a way to indicate that YANG schema nodes represent ephemeral state. It is desirable to allow for, and have to way to indicate, config false YANG schema nodes that are writable operational state.

5. NETCONF Features for Ephemeral State

Ephemeral-REQ-09: The conceptual changes to NETCONF

1. Support for communication mechanisms to enable an I2RS client to determine that an I2RS agent supports the mechanisms needed for I2RS operation.

2. The ephemeral state must support notification of write conflicts using the priority requirements defined in section 7 below in requirements Ephemeral-REQ-11 through Ephemeral-REQ-14).

6. RESTCONF Features for Ephemeral State

Ephemeral-REQ-10: The conceptual changes to RESTCONF are:

1. Support for communication mechanisms to enable an I2RS client to determine that an I2RS agent supports the mechanisms needed for I2RS operation.

2. The ephemeral state must support notification of write conflicts using the priority requirements defined in section 7 below in requirements Ephemeral-REQ-11 through Ephemeral-REQ-14).

7. Requirements regarding Supporting Multi-Head Control via Client Priority

To support Multi-Headed Control, I2RS requires that there be a decidable means of arbitrating the correct state of data when multiple clients attempt to manipulate the same piece of data. This is done via a priority mechanism with the highest priority winning. This priority is per-client.

Ephemeral-REQ-11: The data nodes MAY store I2RS client identity and not the effective priority at the time the data node is stored. Per SEC-REQ-07 in section 3.1 of [I-D.ietf-i2rs-protocol-security-requirements], an identifier must have just one priority. Therefore, the data nodes MAY store I2RS client identity and not the effective priority of the I2RS client at the time the data node is stored. The priority MAY be dynamically changed by AAA, but the exact actions are part of the protocol definition as long as collisions are handled as described in Ephemeral-REQ-12, Ephemeral-REQ-13, and Ephemeral-REQ-14.

Ephemeral-REQ-12: When a collision occurs as two clients are trying to write the same data node, this collision is considered an error and priorities were created to give a deterministic result. When there is a collision, a notification (which includes indicating data node the collision occurred on) MUST BE sent to the original client to give the original client a chance to deal with the issues surrounding the collision. The original client may need to fix their state.

Note:RESTCONF and NETCONF posts can come in concurrently from alternative sources (see ETag in [I-D.ietf-netconf-restconf] section 3.4.1.2 usage). Therefore the collision detection and comparison of priority needs to occur both for both type of updates (POST or edit-config) at the point of comparison.

Ephemeral-REQ-13: Multi-headed control is required for collisions and the priority resolution of collisions. Multi-headed control is not tied to ephemeral state. I2RS is not mandating how AAA supports priority. Mechanisms which prevent collisions of two clients trying to modify the same node of data are the focus.

Ephemeral-REQ-14: A deterministic conflict resolution mechanism MUST be provided to handle the error scenario that two clients, with the same priority, update the same configuration data node. The I2RS architecture gives one way that this could be achieved, by specifying that the first update wins. Other solutions, that prevent oscillation of the config data node, are also acceptable.

8. Multiple Message Transactions

Ephemeral-REQ-15: Section 7.9 of the [I-D.ietf-i2rs-architecture] states the I2RS architecture does not include multi-message atomicity and roll-back mechanisms. I2RS notes multiple operations in one or more messages handling can handle errors within the set of operations in many ways. No multi-message commands SHOULD cause errors to be inserted into the I2RS ephemeral state.

9. Pub/Sub Requirements Expanded for Ephemeral State

I2RS clients require the ability to monitor changes to ephemeral state. While subscriptions are well defined for receiving notifications, the need to create a notification set for all ephemeral configuration state may be overly burdensome to the user.

There is thus a need for a general subscription mechanism that can provide notification of changed state, with sufficient information to permit the client to retrieve the impacted nodes. This should be doable without requiring the notifications to be created as part of every single I2RS module.

The publication/subscription requirements for I2RS are in [I-D.ietf-i2rs-pub-sub-requirements], and the following general requirements SHOULD be understood to be expanded to to include ephemeral state:

- Pub-Sub-REQ-01: The Subscription Service MUST support subscriptions against ephemeral state in operational data stores, configuration data stores or both.

- Pub-Sub-REQ-02: The Subscription Service MUST support filtering so that subscribed updates under a target node might publish only ephemeral state in operational data or configuration data, or publish both ephemeral and operational data.

- Pub-Sub-REQ-03: The subscription service must support subscriptions which are ephemeral. (E.g. An ephemeral data model which has ephemeral subscriptions.)

10. IANA Considerations

There are no IANA requirements for this document.

11. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements] document. The security requirements for the I2RS protocol environment are in [I-D.ietf-i2rs-security-environment-reqs].

12. Acknowledgements

This document is an attempt to distill lengthy conversations on the I2RS mailing list for an architecture that was for a long period of time a moving target. Some individuals in particular warrant specific mention for their extensive help in providing the basis for this document:

- Alia Atlas
- Andy Bierman
- Martin Bjorklund
- Dean Bogdanavich
- Rex Fernando
- Joel Halpern
- Thomas Nadeau
- Juergen Schoenwaelder
- Kent Watsen
- Robert Wilton

13. References

13.1. Normative References:

[I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-15 (work in progress), April 2016.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-06 (work in progress), May 2016.

[I-D.ietf-i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-09 (work in progress), May 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-01 (work in progress), April 2016.

[I-D.ietf-i2rs-traceability]
Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", draft-ietf-i2rs-traceability-11 (work in progress), May 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-14 (work in progress), June 2016.

[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>.

13.2. Informative References

[I-D.hares-i2rs-protocol-strawman]
Hares, S., Bierman, A., and a. amit.dass@ericsson.com, "I2RS protocol strawman", draft-hares-i2rs-protocol-strawman-02 (work in progress), May 2016.

[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>.

Authors' Addresses

Jeff Haas
Juniper

Email: jhaas@juniper.net

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

D. Bogdanovic
Volta Networks
K. Sreenivasa
Cisco Systems
L. Huang
General Electric
D. Blair
Cisco Systems
July 08, 2016

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-08

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 January 9, 2017.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

The match criteria consist of a tuple of packet header match criteria and can have metadata match criteria as well.

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

- o In case vendor suppors it, 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 knowns 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

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. It as also possible for ACE to match on metadata, if supported by the vendor. 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]
         +--rw acl-name               string
         +--rw acl-type               acl-type
         +--ro acl-oper-data
         +--rw access-list-entries
            +--rw ace* [rule-name]
               +--rw rule-name        string
               +--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 actions
               |  +--rw (packet-handling)?
               |     +--:(deny)
               |     |  +--rw deny?     empty
               |     +--:(permit)
               |        +--rw permit?   empty
               +--ro ace-oper-data
                  +--ro match-counter?   yang:counter64
```

Figure 1

4. ACL YANG Models

4.1. IETF Access Contorl 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@2016-07-08.yang"

```
module ietf-access-control-list {
  yang-version 1.1;
  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@cisco.com
    Editor: Lisa Huang
    lyihuang16@gmail.com
    Editor: Dana Blair
    dblair@cisco.com";
  description
    "This YANG module defines a component that describing the
    configuration of Access Control Lists (ACLs).
    Copyright (c) 2016 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 2016-07-08 {
      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,
        like 10/100/1000baseT or WiFi Access Control List. 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.";
        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.";
        }
        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)";
            leaf rule-name {
              type string;
              description
                "A unique name identifying this Access List
                Entry(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 Acess 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;
                }
              }
            }
            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";
              }
            }
          }
        }
      }
    }
  }
  <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, and transport layer fields. 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@2016-07-08.yang"
module ietf-packet-fields {
  yang-version 1.1;
  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@cisco.com
    Editor: Lisa Huang
    lyihuang16@gmail.com
    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) 2016 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 2016-07-08 {
    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 {
        type inet:port-number;
        must ". >= ../lower-port" {
          error-message
          "The upper-port must be greater than or equal to lower-port";
        }
        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 {
        type inet:port-number;
        must ". >= ../lower-port" {
          error-message
            "The upper-port must be greater than or equal to lower-port";
        }

        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.";
  }

}
<CODE ENDS>
```

4.3. An ACL Example

Requirement: Deny All traffic from 10.10.10.1/24.

Here is the acl configuration xml for this Access Control List:

```
<?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-name>sample-ipv4-acl</acl-name>
        <acl-type>ipv4-acl</acl-type>
        <access-list-entries>
          <ace>
            <rule-name>rule1</rule-name>
            <matches>
              <source-ipv4-network>
                10.10.10.1/24
              </source-ipv4-network>
            </matches>
            <actions>
              <deny />
            </actions>
          </ace>
        </access-list-entries>
      </acl>
    </access-lists>
  </data>
```

The acl and aces can be described in CLI as the following:

```
      access-list ip sample-ip-acl
      deny tcp 10.10.10.1/24 any
```

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 filewall 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.

The example in Section 4.3 can be configured using nftable tool as below.

```
nft add table ip filter
nft add chain filter input
nft add rule ip filter input ip protocol tcp ip saddr 10.10.10.1/24 drop
```

The configuration entries added in nftable would be.

```
table ip filter {
  chain input {
    ip protocol tcp ip saddr 10.10.10.1/24 drop
  }
}
```

We can see that there are many similarities between Linux nftables and IETF ACL YANG data models and its extension 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 "example-ext-route-filter@2016-02-18.yang"
module example-ext-route-filter {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:example-ext-route-filter";
  prefix example-ext-route-filter;
  import ietf-inet-types {
    prefix "inet";
  }
  import ietf-access-control-list {
    prefix "ietf-acl";
  }

  organization
    "Route model 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 2016-02-18 {
       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 two new choices, protocol-payload-choice and metadata. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. Metadata matches apply to fields associated with the packet but not in the packet header such as input interface or overall packet length. 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
   +--rw (metadata)?
      +--:(interface-name)
         +--rw interface-name* [input-interface]
            +--rw input-interface    if:interface-ref
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 hierarchitacl-policer?   string
augment /ietf-acl:access-lists/ietf-acl:acl:
   +--rw default-actions
      +--rw deny?   empty


file "newco-acl@2016-07-08.yang"
module example-newco-acl {
  yang-version 1.1;

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

  prefix example-newco-acl;

  import ietf-access-control-list {
    prefix "ietf-acl";
  }
  import ietf-interfaces {
    prefix if;
  }
  organization
    "Newco model group.";

  contact
    "abc@newco.com";
  description
    "This YANG module augment IETF ACL Yang.";

  revision 2016-07-08{
```

```
    description
      "Creating NewCo proprietary extensions to ietf-acl model";
    reference
      "RFC XXXX: Network Access Control List (ACL)
      YANG Data  Model";
  }

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

    choice metadata {
      description "Newco proprietary interface match condition";
      list interface-name {
        key input-interface;
        ordered-by user;
        description "Match interface name";
        uses metadata;
      }
    }
  }

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

```
      case hiearchical-policer {
        description "Name of hierarchical policer to use to
        rate-limit traffic";
        leaf hierarchitacl-policer{
          type string;
          description "";
        }
      }
    }
  }

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

  grouping metadata {
    description
    "Fields associated with a packet which are not in
    the header.";
    leaf input-interface {
      type if:interface-ref {
        require-instance false;
      }
      description
        "Packet was received on this interface";
    }
  }

  grouping match-simple-payload-protocol-value {
    description "Newco proprietary payload";
    leaf value-keyword {
      type enumeration {
        enum icmp {
          description "Internet Control Message Protocol";
        }
        enum icmp6 {
          description "Internet Control Message Protocol Version 6";
        }
        enum range {
          description "Range of values";
```

```
        }
      }

      description "(null)";
    }
  }
}
```

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 involved 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";
  }
  import example-newco-acl {
    prefix "example-newco";
    }

  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
Volta Networks

Email: ivandean@gmail.com

Kiran Agrahara Sreenivasa
Cisco Systems

Email: kkoushik@cisco.com

Lisa Huang
General Electric

Email: lyihuang16@gmail.com

Dana Blair
Cisco Systems

Email: dblair@cisco.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks

July 7, 2016

Common Interface Extension YANG Data Models
draft-ietf-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.

Table of Contents

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:

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

2. Objectives

The aim of 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:

- A bandwidth configuration leaf to specify the bandwidth available on an interface to control routing metrics.
- A carrier delay feature used to provide control over short lived link state flaps.
- An interface link state dampening feature that is used to provide control over longer lived link state flaps.
- An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.

- A loopback configuration leaf that is primarily aimed at loopback at the physical layer.

- MTU configuration leaves applicable to all packet/frame based interfaces.

- A transport layer leaf to indicate whether the interface handles traffic at L1, L2 or L3.

- 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: ietf-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 check sequence (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: ietf-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 "ietf-interfaces-common@2016-07-07.yang"
module ietf-interfaces-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces-common";
  prefix if-cmn;
```

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

  import iana-if-type {
    prefix ianaift;
  }

  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: Lou Berger
               <mailto:lberger@labn.net>

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

     Editor:   Robert Wilton
               <mailto:rwilton@cisco.com>";

  description
    "This module contains common definitions for extending the IETF
     interface YANG model (RFC 7223) with common configurable layer 2
     properties.

     Copyright (c) 2016 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 XXX; see the RFC
     itself for full legal notices.";

  revision 2016-07-07 {
    description
      "Update module title and description text to IETF standard
       text";

    reference "Internet draft: draft-ietf-netmod-intf-ext-yang-01";
```

```
  }

  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
               capabilties 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, '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 tranport 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 "ietf-interfaces-ethernet-like@2016-07-07.yang"

```
   module ietf-interfaces-ethernet-like {
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like";

     prefix ethlike;

     import ietf-interfaces {
       prefix if;
     }

     import ietf-yang-types {
       prefix yang;
     }

     import iana-if-type {
       prefix ianaift;
     }

     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: Lou Berger
                  <mailto:lberger@labn.net>

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

        Editor:   Robert Wilton
                  <mailto: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.

        Copyright (c) 2016 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 XXX; see the RFC
        itself for full legal notices.";

      revision 2016-07-07 {
        description
          "Update module title and description text to IETF standard
           text";

        reference "Internet draft: draft-ietf-netmod-intf-ext-yang-01";
      }

      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:

- /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 excesssive traffic on a particular interface or network device:

- /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:

- /if:interfaces/if:interface/carrier-delay/down

- /if:interfaces/if:interface/carrier-delay/up

- /if:interfaces/if:interface/dampening/half-life
- /if:interfaces/if:interface/dampening/reuse
- /if:interfaces/if:interface/dampening/suppress
- /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:

- /if:interfaces/if:interface/encapsulation
- /if:interfaces/if:interface/l2-mtu
- /if:interfaces/if:interface/l3-mtu
- /if:interfaces/if:interface/transport-layer

Normally devices will not allow the parent-interface leaf to be changed after the interfce 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:

- /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:

- 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
Cisco Systems

Email: tapsingh@juniper.net

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Network Working Group
Internet-Draft
Obsoletes: 6087 (if approved)
Intended status: BCP
Expires: September 14, 2018

A. Bierman
YumaWorks

March 13, 2018

Guidelines for Authors and Reviewers of YANG Data Model Documents
draft-ietf-netmod-rfc6087bis-20

Abstract

This memo provides guidelines for authors and reviewers of specifications containing YANG data model modules. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) and RESTCONF protocol implementations that utilize YANG data model modules. This document obsoletes RFC 6087.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 14, 2018.

Copyright Notice

Table of Contents

1. Introduction

The standardization of network configuration interfaces for use with network configuration management protocols, such as the Network Configuration Protocol [RFC6241] and RESTCONF [RFC8040], requires a modular set of data models, which can be reused and extended over time.

This document defines a set of usage guidelines for documents containing YANG 1.1 [RFC7950] and YANG 1.0 [RFC6020] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF and/or RESTCONF server. A NETCONF or RESTCONF server that supports a particular YANG module will support client NETCONF and/or RESTCONF operation requests, as indicated by the specific content defined in the YANG module.

Many YANG constructs are defined as optional to use, such as the description statement. However, in order to make YANG modules more useful, it is desirable to define a set of usage guidelines that entails a higher level of compliance than the minimum level defined in the YANG specification.

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

This document defines usage guidelines related to the NETCONF operations layer and NETCONF content layer, as defined in [RFC6241], and the RESTCONF methods and RESTCONF resources, as defined in [RFC8040],

These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.

Note that this document is not a YANG tutorial and the reader is expected to know the YANG data modeling language before using this document.

1.1. Changes Since RFC 6087

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

- Updated NETCONF reference from RFC 4741 to RFC 6241

- Updated NETCONF over SSH citation from RFC 4742 to RFC 6242
- Updated YANG Types reference from RFC 6021 to RFC 6991
- Updated obsolete URLs for IETF resources
- Changed top-level data node guideline
- Clarified XPath usage for a literal value representing a YANG identity
- Clarified XPath usage for a when-stmt
- Clarified XPath usage for 'proceeding-sibling' and 'following-sibling' axes
- Added terminology guidelines
- Added YANG tree diagram guidelines
- Updated XPath guidelines for type conversions and function library usage.
- Updated data types section
- Updated notifications section
- Clarified conditional key leaf nodes
- Clarify usage of 'uint64' and 'int64' data types
- Added text on YANG feature usage
- Added Identifier Naming Conventions
- Clarified use of mandatory nodes with conditional augmentations
- Clarified namespace and domain conventions for example modules
- Clarified conventions for identifying code components
- Added YANG 1.1 guidelines
- Added Data Model Constraints section
- Added mention of RESTCONF protocol

- Added guidelines for NMDA Revised Datastores

2. Terminology

2.1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. NETCONF Terms

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

- capabilities
- client
- operation
- server

2.3. YANG Terms

The following terms are defined in [RFC7950] and are not redefined here:

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

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

2.4. NMDA Terms

The following terms are defined in the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores]. and are not redefined here:

- configuration
- conventional configuration datastore
- datastore
- operational state
- operational state datastore

2.5. Terms

The following terms are used throughout this document:

- published: A stable release of a module or submodule. For example the "Request for Comments" described in section 2.1 of [RFC2026] is considered a stable publication.
- unpublished: An unstable release of a module or submodule. For example the "Internet-Draft" described in section 2.2 of [RFC2026] is considered an unstable publication that is a work-in-progress, subject to change at any time.
- YANG fragment: A set of YANG statements that are not intended to represent a complete YANG module or submodule. These statements are not intended for actual use, except to provide an example of YANG statement usage. The invalid syntax "..." is sometimes used to indicate that additional YANG statements would be present in a real YANG module.
- YANG tree diagram: a diagram representing the contents of a YANG module, as defined in [I-D.ietf-netmod-yang-tree-diagrams]. Also called a "tree diagram".

3. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors [ID-Guidelines] MUST be followed. The RFC Editor provides guidelines for authors of RFCs, which are first published as Internet-Drafts. These guidelines should be followed and are defined in [RFC7322] and updated in [RFC7841], "RFC Document Style" [RFC-STYLE], and [I-D.flanagan-7322bis].

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

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

There are three usage scenarios for YANG that can appear in an Internet-Draft or RFC:

- normative module or submodule
- example module or submodule
- example YANG fragment not part of any module or submodule

The guidelines in this document refer mainly to a normative module or submodule, but may be applicable to example modules and YANG fragments as well.

3.1. Module Copyright

The module description statement MUST contain a reference to the latest approved IETF Trust Copyright statement, which is available online at:

https://trustee.ietf.org/license-info/

3.2. Code Components

Each normative YANG module or submodule contained within an Internet-Draft or RFC is considered to be a code component. The strings "<CODE BEGINS>" and "<CODE ENDS>" MUST be used to identify each code component.

The "<CODE BEGINS>" tag SHOULD be followed by a string identifying the file name specified in Section 5.2 of [RFC7950]. The name string form that includes the revision-date SHOULD be used. The revision date MUST match the date used in the most recent revision of the module.

The following example is for the '2016-03-20' revision of the 'ietf-foo' module:

```
<CODE BEGINS> file "ietf-foo@2016-03-20.yang"

    module ietf-foo {
      namespace "urn:ietf:params:xml:ns:yang:ietf-foo";
      prefix "foo";
      organization "...";
      contact "...";
      description "...";
      revision 2016-03-20 {
        description "Latest revision";
        reference "RFC XXXX: Foo Protocol";
      }
      // ... more statements
    }

<CODE ENDS>
```

3.2.1. Example Modules

Example modules are not code components. The <CODE BEGINS> convention MUST NOT be used for example modules.

An example module SHOULD be named using the term "example", followed by a hyphen, followed by a descriptive name, e.g., "example-toaster". See Section 4.9 regarding the namespace guidelines for example modules.

3.3. Terminology Section

A terminology section MUST be present if any terms are defined in the document or if any terms are imported from other documents.

3.4. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module, and SHOULD be included to help readers understand YANG module structure. Guidelines on tree diagrams can be found in Section 3 of [I-D.ietf-netmod-yang-tree-diagrams].

If YANG tree diagrams are used, then an informative reference to the YANG tree diagrams specification MUST be included in the document. Refer to Section 2.2 of [I-D.ietf-netmod-rfc8022bis] for an example of such a reference.

3.5. Narrative Sections

The narrative part MUST include an overview section that describes the scope and field of application of the module(s) defined by the specification and that specifies the relationship (if any) of these modules to other standards, particularly to standards containing other YANG modules. The narrative part SHOULD include one or more sections to briefly describe the structure of the modules defined in the specification.

If the module(s) defined by the specification imports definitions from other modules (except for those defined in the [RFC7950] or [RFC6991] documents), or are always implemented in conjunction with other modules, then those facts MUST be noted in the overview section, as MUST be noted any special interpretations of definitions in other modules. Refer to section 2.3 of [I-D.ietf-netmod-rfc8022bis] for an example of this overview section.

If the documents contains YANG module(s) that are compliant with the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores], then the Introduction section should mention this fact.

Example:

> The YANG model in this document conforms to the Network Management Datastore Architecture defined in [I-D.ietf-netmod-revised-datastores].

Consistent indentation SHOULD be used for all examples, including YANG fragments and protocol message instance data. If line wrapping is done for formatting purposes, then this SHOULD be noted, as shown in the following example:

```
   [note: '\' line wrapping for formatting only]
```

```
<myleaf xmlns="tag:example.com,2017:example-two">\
  this is a long value so the line needs to wrap to stay\
  within 72 characters\
</myleaf>
```

3.6. Definitions Section

This section contains the module(s) defined by the specification. These modules SHOULD be written using the YANG 1.1 [RFC7950] syntax. YANG 1.0 [RFC6020] syntax MAY be used if no YANG 1.1 constructs or semantics are needed in the module. If any of the imported YANG modules are written using YANG 1.1, then the module MUST be written using YANG 1.1.

A YIN syntax version of the module MAY also be present in the document. There MAY also be other types of modules present in the document, such as SMIv2, which are not affected by these guidelines.

Note that if the module itself is considered normative and not an example module or example YANG fragment, then all YANG statements within a YANG module are considered normative. The use of keywords defined in [RFC2119] apply to YANG description statements in normative modules exactly as they would in any other normative section.

Example YANG modules and example YANG fragments MUST NOT contain any normative text, including any all-uppercase reserved words from [RFC2119].

Consistent indentation and formatting SHOULD be used in all YANG statements within a module.

See Section 4 for guidelines on YANG usage.

3.7. Security Considerations Section

Each specification that defines one or more modules MUST contain a section that discusses security considerations relevant to those modules.

This section MUST be patterned after the latest approved template (available at https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines). Section 3.7.1 contains the security considerations template dated 2013-05-08 and last updated 2017-12-21. Authors MUST check the WEB page at the URL listed above in case there is a more recent version available.

In particular:

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

- Readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

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

3.7.1. Security Considerations Section Template

```
X. Security Considerations

The YANG module specified in this document defines a schema for data
that is designed to be accessed via network management protocols such
as NETCONF [RFC6241] or RESTCONF [RFC8040].  The lowest NETCONF layer
is the secure transport layer, and the mandatory-to-implement secure
transport is Secure Shell (SSH) [RFC6242].  The lowest RESTCONF layer
is HTTPS, and the mandatory-to-implement secure transport is TLS
[RFC5246].

The NETCONF access control model [RFC6536] provides the means to
restrict access for particular NETCONF or RESTCONF users to a
preconfigured subset of all available NETCONF or RESTCONF protocol
operations and content.

 -- if you have any writeable data nodes (those are all the
 -- "config true" nodes, and remember, that is the default)
 -- describe their specific sensitivity or vulnerability.

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

<list subtrees and data nodes and state why they are sensitive>
```

```
    -- for all YANG modules you must evaluate whether any readable data
    -- nodes (those are all the "config false" nodes, but also all other
    -- nodes, because they can also be read via operations like get or
    -- get-config) are sensitive or vulnerable (for instance, if they
    -- might reveal customer information or violate personal privacy
    -- laws such as those of the European Union if exposed to
    -- unauthorized parties)
```

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

<list subtrees and data nodes and state why they are sensitive>

```
    -- if your YANG module has defined any rpc operations
    -- describe their specific sensitivity or vulnerability.
```

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

<list RPC operations and state why they are sensitive>

3.8. IANA Considerations Section

In order to comply with IESG policy as set forth in https://www.ietf.org/id-info/checklist.html, every Internet-Draft that is submitted to the IESG for publication MUST contain an IANA Considerations section. The requirements for this section vary depending on what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section stating that there are no actions might be removed by the RFC Editor before publication. Refer to the guidelines in [RFC8126] for more details.

Each normative YANG module MUST be registered in the XML namespace Registry [RFC3688], and the YANG Module Names Registry [RFC6020]. This applies to new modules and updated modules. Examples of these registrations for the "ietf-template" module can be found in Section 5.

3.8.1. Documents that Create a New Namespace

If an Internet-Draft defines a new namespace that is to be administered by the IANA, then the document MUST include an IANA

Considerations section that specifies how the namespace is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new YANG Namespace registry entry MUST be requested from the IANA. The [RFC7950] specification includes the procedure for this purpose in its IANA Considerations section.

3.8.2. Documents that Extend an Existing Namespace

It is possible to extend an existing namespace using a YANG submodule that belongs to an existing module already administered by IANA. In this case, the document containing the main module MUST be updated to use the latest revision of the submodule.

3.9. Reference Sections

For every import or include statement that appears in a module contained in the specification, that identifies a module in a separate document, a corresponding normative reference to that document MUST appear in the Normative References section. The reference MUST correspond to the specific module version actually used within the specification.

For every normative reference statement that appears in a module contained in the specification, that identifies a separate document, a corresponding normative reference to that document SHOULD appear in the Normative References section. The reference SHOULD correspond to the specific document version actually used within the specification. If the reference statement identifies an informative reference, that identifies a separate document, a corresponding informative reference to that document MAY appear in the Informative References section.

3.10. Validation Tools

All modules need to be validated before submission in an Internet Draft. The 'pyang' YANG compiler is freely available from github:

https://github.com/mbj4668/pyang

If the 'pyang' compiler is used to validate a normative module, then the "--ietf" command line option MUST be used to identify any IETF guideline issues.

If the 'pyang' compiler is used to validate an example module, then the "--ietf" command line option MAY be used to identify any IETF guideline issues.

The "yanglint" program is also freely available from github.

https://github.com/CESNET/libyang

This tool can be used to validate XPath statements within YANG modules.

3.11. Module Extraction Tools

A version of 'rfcstrip' is available which will extract YANG modules from an Internet Draft or RFC. The 'rfcstrip' tool which supports YANG module extraction is freely available:

http://www.yang-central.org/twiki/pub/Main/YangTools/rfcstrip

This tool can be used to verify that the "<CODE BEGINS>" and "<CODE ENDS>" tags are used correctly and that the normative YANG modules can be extracted correctly.

The "xym" tool is freely available on github and can be used to extract YANG modules from a document.

https://github.com/xym-tool/xym

3.12. Module Usage Examples

Each specification that defines one or more modules SHOULD contain usage examples, either throughout the document or in an appendix. This includes example instance document snippets in an appropriate encoding (e.g., XML and/or JSON) to demonstrate the intended usage of the YANG module(s). Example modules MUST be validated. Refer to Section 3.10 for tools which validate YANG modules. If IP addresses are used, then either a mix of IPv4 and IPv6 addresses or IPv6 addresses exclusively SHOULD be used in the examples.

4. YANG Usage Guidelines

Modules in IETF Standards Track specifications MUST comply with all syntactic and semantic requirements of YANG 1.1 [RFC7950]. See the exception for YANG 1.0 in Section 3.6. The guidelines in this section are intended to supplement the YANG specification, which is intended to define a minimum set of conformance requirements.

In order to promote interoperability and establish a set of practices based on previous experience, the following sections establish usage guidelines for specific YANG constructs.

Only guidelines that clarify or restrict the minimum conformance requirements are included here.

4.1. Module Naming Conventions

Normative modules contained in Standards Track documents MUST be named according to the guidelines in the IANA Considerations section of [RFC7950].

A distinctive word or acronym (e.g., protocol name or working group acronym) SHOULD be used in the module name. If new definitions are being defined to extend one or more existing modules, then the same word or acronym should be reused, instead of creating a new one.

All published module names MUST be unique. For a YANG module published in an RFC, this uniqueness is guaranteed by IANA. For unpublished modules, the authors need to check that no other work in progress is using the same module name.

Example modules are non-normative, and SHOULD be named with the prefix "example-".

It is suggested that a stable prefix be selected representing the entire organization. All normative YANG modules published by the IETF MUST begin with the prefix "ietf-". Another standards organization, such as the IEEE, might use the prefix "ieee-" for all YANG modules.

Once a module name is published, it MUST NOT be reused, even if the RFC containing the module is reclassified to ’Historic’ status. A module name cannot be changed in YANG, and this would be treated as a a new module, not a name change.

4.2. Prefixes

All YANG definitions are scoped by the module containing the definition being referenced. This allows definitions from multiple modules to be used, even if the names are not unique. In the example below, the identifier "foo" is used in all 3 modules:

```
module example-foo {
  namespace "tag:example.com,2017:example-foo";
  prefix f;

  container foo;
}

module example-bar {
  namespace "tag:example.com,2017:example-bar";
  prefix b;

  typedef foo { type uint32; }
}

module example-one {
  namespace "tag:example.com,2017:example-one";
  prefix one;
  import example-foo { prefix f; }
  import example-bar { prefix b; }

  augment "/f:foo" {
     leaf foo { type b:foo; }
  }
}
```

YANG defines the following rules for prefix usage:

- Prefixes are never used for built in data types and YANG keywords.
- A prefix MUST be used for any external statement (i.e., a statement defined with the YANG "extension" statement)
- The proper module prefix MUST be used for all identifiers imported from other modules
- The proper module prefix MUST be used for all identifiers included from a submodule.

The following guidelines apply to prefix usage of the current (local) module:

- The local module prefix SHOULD be used instead of no prefix in all path expressions.

- The local module prefix MUST be used instead of no prefix in all "default" statements for an "identityref" or "instance-identifier" data type

- The local module prefix MAY be used for references to typedefs, groupings, extensions, features, and identities defined in the module.

Prefix values SHOULD be short, but also likely to be unique. Prefix values SHOULD NOT conflict with known modules that have been previously published.

4.3. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an 'identifier-arg-str' token in the ABNF in Section 13 of [RFC7950].

4.3.1. Identifier Naming Conventions

Identifiers SHOULD follow a consistent naming pattern throughout the module. Only lower-case letters, numbers, and dashes SHOULD be used in identifier names. Upper-case characters, the period character, and the underscore character MAY be used if the identifier represents a well-known value that uses these characters. YANG does not permit any other characters in YANG identifiers.

Identifiers SHOULD include complete words and/or well-known acronyms or abbreviations. Child nodes within a container or list SHOULD NOT replicate the parent identifier. YANG identifiers are hierarchical and are only meant to be unique within the the set of sibling nodes defined in the same module namespace.

It is permissible to use common identifiers such as "name" or "id" in data definition statements, especially if these data nodes share a common data type.

Identifiers SHOULD NOT carry any special semantics that identify data modelling properties. Only YANG statements and YANG extension statements are designed to convey machine readable data modelling properties. For example, naming an object "config" or "state" does not change whether it is configuration data or state data. Only defined YANG statements or YANG extension statements can be used to assign semantics in a machine readable format in YANG.

4.4. Defaults

In general, it is suggested that substatements containing very common default values SHOULD NOT be present. The following substatements are commonly used with the default value, which would make the module difficult to read if used everywhere they are allowed.

Statement	Default Value
config	true
mandatory	false
max-elements	unbounded
min-elements	0
ordered-by	system
status	current
yin-element	false

Statement Defaults

4.5. Conditional Statements

A module may be conceptually partitioned in several ways, using the 'if-feature' and/or 'when' statements.

Data model designers need to carefully consider all modularity aspects, including the use of YANG conditional statements.

If a data definition is optional, depending on server support for a NETCONF or RESTCONF protocol capability, then a YANG 'feature' statement SHOULD be defined. The defined "feature" statement SHOULD then be used in the conditional "if-feature" statement referencing the optional data definition.

If any notification data, or any data definition, for a non-configuration data node is not mandatory, then the server may or may not be required to return an instance of this data node. If any conditional requirements exist for returning the data node in a notification payload or retrieval request, they MUST be documented somewhere. For example, a 'when' or 'if-feature' statement could apply to the data node, or the conditional requirements could be explained in a 'description' statement within the data node or one of its ancestors (if any).

If any 'if-feature' statements apply to a list node, then the same 'if-feature' statements MUST apply to any key leaf nodes for the list. There MUST NOT be any 'if-feature' statements applied to any

key leaf that do not also apply to the parent list node.

There SHOULD NOT be any 'when' statements applied to a key leaf node. It is possible that a 'when' statement for an ancestor node of a key leaf will have the exact node-set result as the key leaf. In such a case, the 'when' statement for the key leaf is redundant and SHOULD be avoided.

4.6. XPath Usage

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

4.6.1. XPath Evaluation Contexts

YANG defines 5 separate contexts for evaluation of XPath statements:

1) The "running" datastore: collection of all YANG configuration data nodes. The document root is the conceptual container, (e.g., "config" in the "edit-config" operation), which is the parent of all top-level data definition statements with a "config" statement value of "true".

2) State data + the "running" datastore: collection of all YANG data nodes. The document root is the conceptual container, parent of all top-level data definition statements.

3) Notification: an event notification document. The document root is the notification element.

4) RPC Input: The document root is the conceptual "input" node, which is the parent of all RPC input parameter definitions.

5) RPC Output: The document root is the conceptual "output" node, which is the parent of all RPC output parameter definitions.

Note that these XPath contexts cannot be mixed. For example, a "when" statement in a notification context cannot reference configuration data.

```
      notification foo {
        leaf mtu {
          // NOT OK because when-stmt context is this notification
          when "/if:interfaces/if:interface[name='eth0']";
          type leafref {
            // OK because path-stmt has a different context
            path "/if:interfaces/if:interface/if:mtu";
          }
        }
      }
```

It is especially important to consider the XPath evaluation context for XPath expressions defined in groupings. An XPath expression defined in a grouping may not be portable, meaning it cannot be used in multiple contexts and produce proper results.

If the XPath expressions defined in a grouping are intended for a particular context, then this context SHOULD be identified in the "description" statement for the grouping.

4.6.2. Function Library

The 'position' and 'last' functions SHOULD NOT be used. This applies to implicit use of the 'position' function as well (e.g., '//chapter[42]'). A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'position' and 'last' functions MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

The 'id' function SHOULD NOT be used. The 'ID' attribute is not present in YANG documents so this function has no meaning. The YANG compiler SHOULD return an empty string for this function.

The 'namespace-uri' and 'name' functions SHOULD NOT be used. Expanded names in XPath are different than YANG. A specific canonical representation of a YANG expanded name does not exist.

The 'lang' function SHOULD NOT be used. This function does not apply to YANG because there is no 'lang' attribute set with the document. The YANG compiler SHOULD return 'false' for this function.

The 'local-name', 'namespace-uri', 'name', 'string', and 'number' functions SHOULD NOT be used if the argument is a node-set. If so, the function result will be determined by the document order of the node-set. Since this order can be different on each server, the function results can also be different. Any function call that implicitly converts a node-set to a string will also have this issue.

The 'local-name' function SHOULD NOT be used to reference local names outside of the YANG module defining the must or when expression containing the 'local-name' function. Example of a local-name function that should not be used:

```
/*[local-name()='foo']
```

The 'derived-from-or-self' function SHOULD be used instead of an equality expression for identityref values. This allows the identities to be conceptually augmented.

Example:

```
// do not use
when "md-name-format = 'name-format-null'";

// this is preferred
when "derived-from-or-self(md-name-format, 'name-format-null')";
```

4.6.3. Axes

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

The 'preceding', and 'following' axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF or RESTCONF server configuration database, which may not be supported consistently or produce reliable results across implementations. Predicate expressions based on static node properties (e.g., element name or value, 'ancestor' or 'descendant' axes) SHOULD be used instead. The 'preceding' and 'following' axes MAY be used if document order is not relevant to the outcome of the expression (e.g., check for global uniqueness of a parameter value).

The 'preceding-sibling' and 'following-sibling' axes SHOULD NOT used, however they MAY be used if document order is not relevant to the outcome of the expression.

A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'preceding-sibling' and 'following-sibling' axes MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

4.6.4. Types

Data nodes that use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric or boolean expressions. There are boundary

conditions in which the translation from the YANG 64-bit type to an XPath number can cause incorrect results. Specifically, an XPath 'double' precision floating point number cannot represent very large positive or negative 64-bit numbers because it only provides a total precision of 53 bits. The 'int64' and 'uint64' data types MAY be used in numeric expressions if the value can be represented with no more than 53 bits of precision.

Data modelers need to be careful not to confuse the YANG value space and the XPath value space. The data types are not the same in both, and conversion between YANG and XPath data types SHOULD be considered carefully.

Explicit XPath data type conversions MAY be used (e.g., 'string', 'boolean', or 'number' functions), instead of implicit XPath data type conversions.

XPath expressions that contain a literal value representing a YANG identity SHOULD always include the declared prefix of the module where the identity is defined.

XPath expressions for 'when' statements SHOULD NOT reference the context node or any descendant nodes of the context node. They MAY reference descendant nodes if the 'when' statement is contained within an 'augment' statement, and the referenced nodes are not defined within the 'augment' statement.

Example:

```
augment "/rt:active-route/rt:input/rt:destination-address" {
   when "rt:address-family='v4ur:ipv4-unicast'" {
     description
       "This augment is valid only for IPv4 unicast.";
   }
   // nodes defined here within the augment-stmt
   // cannot be referenced in the when-stmt
}
```

4.6.5. Wildcards

It is possible to construct XPath expressions that will evaluate differently when combined with several modules within a server implementation, then when evaluated within the single module. This is due to augmenting nodes from other modules.

Wildcard expansion is done within a server against all the nodes from all namespaces, so it is possible for a 'must' or 'when' expression that uses the '*' operator will always evaluate to false if processed

within a single YANG module. In such cases, the 'description' statement SHOULD clarify that augmenting objects are expected to match the wildcard expansion.

```
   when /foo/services/*/active {
     description
       "No services directly defined in this module.
        Matches objects that have augmented the services container.";
   }
```

4.6.6. Boolean Expressions

The YANG "must" and "when" statements use an XPath boolean expression to define the test condition for the statement. It is important to specify these expressions in a way that will not cause inadvertent changes in the result if the objects referenced in the expression are updated in future revisions of the module.

For example, the leaf "foo2" must exist if the leaf "foo1" is equal to "one" or "three":

```
     leaf foo1 {
       type enumeration {
          enum one;
          enum two;
          enum three;
       }
     }

     leaf foo2 {
       // INCORRECT
       must "/f:foo1 != 'two'";
       type string;
     }

     leaf foo2 {
       // CORRECT
       must "/f:foo1 = 'one' or /f:foo1 = 'three'";
       type string;
     }
```

In the next revision of the module, leaf "foo1" is extended with a new enum named "four":

```
leaf foo1 {
  type enumeration {
     enum one;
     enum two;
     enum three;
     enum four;
  }
}
```

Now the first XPath expression will allow the enum "four" to be accepted in addition to the "one" and "three" enum values.

4.7. YANG Definition Lifecycle Management

The YANG status statement MUST be present within a definition if its value is 'deprecated' or 'obsolete'. The status SHOULD NOT be changed from 'current' directly to 'obsolete'. An object SHOULD be available for at least one year with 'deprecated' status before it is changed to 'obsolete'.

The module or submodule name MUST NOT be changed, once the document containing the module or submodule is published.

The module namespace URI value MUST NOT be changed, once the document containing the module is published.

The revision-date substatement within the import statement SHOULD be present if any groupings are used from the external module.

The revision-date substatement within the include statement SHOULD be present if any groupings are used from the external submodule.

If an import statement is for a module from a stable source (e.g., an RFC for an IETF module), then a reference-stmt SHOULD be present within an import statement.

```
import ietf-yang-types {
   prefix yang;
   reference "RFC 6991: Common YANG Data Types";
}
```

If submodules are used, then the document containing the main module MUST be updated so that the main module revision date is equal or more recent than the revision date of any submodule that is (directly or indirectly) included by the main module.

Definitions for future use SHOULD NOT be specified in a module. Do not specify placeholder objects like the "reserved" example below:

```
      leaf reserved {
        type string;
        description
          "This object has no purpose at this time, but a future
           revision of this module might define a purpose
           for this object.";
        }
      }
```

4.8. Module Header, Meta, and Revision Statements

For published modules, the namespace MUST be a globally unique URI, as defined in [RFC3986]. This value is usually assigned by the IANA.

The organization statement MUST be present. If the module is contained in a document intended for IETF Standards Track status, then the organization SHOULD be the IETF working group chartered to write the document. For other standards organizations, a similar approach is also suggested.

The contact statement MUST be present. If the module is contained in a document intended for Standards Track status, then the working group web and mailing information SHOULD be present, and the main document author or editor contact information SHOULD be present. If additional authors or editors exist, their contact information MAY be present. There is no need to include the contact information for Working Group chairs.

The description statement MUST be present. For modules published within IETF documents, the appropriate IETF Trust Copyright text MUST be present, as described in Section 3.1.

If the module relies on information contained in other documents, which are not the same documents implied by the import statements present in the module, then these documents MUST be identified in the reference statement.

A revision statement MUST be present for each published version of the module. The revision statement MUST have a reference substatement. It MUST identify the published document that contains the module. Modules are often extracted from their original documents, and it is useful for developers and operators to know how to find the original source document in a consistent manner. The revision statement MAY have a description substatement.

The following example shows the revision statement for a published YANG module:

```
      revision "2012-02-22" {
        description
          "Initial version";
        reference
          "RFC 6536: Network Configuration Protocol (NETCONF)
                     Access Control Model";
      }
```

For an unpublished module, a complete history of each unpublished module revision is not required. That is, within a sequence of draft versions, only the most recent revision need be recorded in the module. Do not remove or reuse a revision statement for a published module. A new revision date is not required unless the module contents have changed. If the module contents have changed, then the revision date of that new module version MUST be updated to a date later than that of the previous version.

The following example shows the two revision statements for an unpublished update to a published YANG module:

```
      revision "2017-12-11" {
        description
          "Added support for YANG 1.1 actions and notifications tied to
           data nodes. Clarify how NACM extensions can be used by other
           data models.";
        reference
          "RFC XXXX: Network Configuration Protocol (NETCONF)
                     Access Control Model";
      }

      revision "2012-02-22" {
        description
          "Initial version";
        reference
          "RFC 6536: Network Configuration Protocol (NETCONF)
                     Access Control Model";
      }
```

4.9. Namespace Assignments

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

- the module to compile correctly instead of generating disruptive fatal errors.

- early implementors to use the modules without picking a random value for the XML namespace.

- early interoperability testing since independent implementations will use the same XML namespace value.

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

A standard namespace statement value SHOULD have the following form:

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

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

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

The following example URNs would be valid namespace statement values for Standards Track modules:

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

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

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

Note that a different URN prefix string SHOULD be used for non-Standards-Track modules. The string SHOULD be selected according to the guidelines in [RFC7950].

The following URIs exemplify what might be used by non Standards Track modules. Note that the domain "example.com" SHOULD be used by example modules in IETF drafts. These URIs are not intended to be de-referenced. They are used for module namespace identification only.

Example URIs using URLs per RFC 3986 [RFC3986]:

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

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

Example URIs using tags per RFC 4151 [RFC4151]:

tag:example.com,2017:example-interfaces

tag:example.com,2017:example-system

4.10. Top-Level Data Definitions

The top-level data organization SHOULD be considered carefully, in advance. Data model designers need to consider how the functionality for a given protocol or protocol family will grow over time.

The separation of configuration data and operational state SHOULD be considered carefully. It is sometimes useful to define separate top-level containers for configuration and non-configuration data. For some existing top-level data nodes, configuration data was not in scope, so only one container representing operational state was created. Refer to the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores] for details.

The number of top-level data nodes within a module SHOULD be minimized. It is often useful to retrieve related information within a single subtree. If data is too distributed, is becomes difficult to retrieve all at once.

The names and data organization SHOULD reflect persistent information, such as the name of a protocol. The name of the working group SHOULD NOT be used because this may change over time.

A mandatory database data definition is defined as a node that a client must provide for the database to be valid. The server is not required to provide a value.

Top-level database data definitions MUST NOT be mandatory. If a mandatory node appears at the top level, it will immediately cause the database to be invalid. This can occur when the server boots or when a module is loaded dynamically at runtime.

4.11. Data Types

Selection of an appropriate data type (i.e., built-in type, existing derived type, or new derived type) is very subjective, and therefore few requirements can be specified on that subject.

Data model designers SHOULD use the most appropriate built-in data type for the particular application.

The signed numeric data types (i.e., 'int8', 'int16', 'int32', and 'int64') SHOULD NOT be used unless negative values are allowed for the desired semantics.

4.11.1. Fixed Value Extensibility

If the set of values is fixed and the data type contents are controlled by a single naming authority, then an enumeration data type SHOULD be used.

```
leaf foo {
  type enumeration {
    enum one;
    enum two;
  }
}
```

If extensibility of enumerated values is required, then the 'identityref' data type SHOULD be used instead of an enumeration or other built-in type.

```
identity foo-type {
  description "Base for the extensible type";
}

identity one {
  base f:foo-type;
}
identity two {
  base f:foo-type;
}

leaf foo {
  type identityref {
    base f:foo-type;
  }
}
```

Note that any module can declare an identity with base "foo-type" that is valid for the "foo" leaf. Identityref values are considered to be qualified names.

4.11.2. Patterns and Ranges

For string data types, if a machine-readable pattern can be defined for the desired semantics, then one or more pattern statements SHOULD be present. A single quoted string SHOULD be used to specify the pattern, since a double-quoted string can modify the content. If the patterns used in a type definition have known limitations such as false negative or false positive matches, then these limitations SHOULD be documented within the typedef or data definition.

The following typedef from [RFC6991] demonstrates the proper use of the "pattern" statement:

```
typedef ipv4-address-no-zone {
  type inet:ipv4-address {
    pattern '[0-9\.]*';
  }
  ...
}
```

For string data types, if the length of the string is required to be bounded in all implementations, then a length statement MUST be present.

The following typedef from [RFC6991] demonstrates the proper use of the "length" statement:

```
typedef yang-identifier {
  type string {
    length "1..max";
    pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
    pattern '.|..|[^xX].*|.[^mM].*|..[^lL].*';
  }
  ...
}
```

For numeric data types, if the values allowed by the intended semantics are different than those allowed by the unbounded intrinsic data type (e.g., 'int32'), then a range statement SHOULD be present.

The following typedef from [RFC6991] demonstrates the proper use of the "range" statement:

```
typedef dscp {
  type uint8 {
     range "0..63";
  }
  ...
}
```

4.11.3. Enumerations and Bits

For 'enumeration' or 'bits' data types, the semantics for each 'enum' or 'bit' SHOULD be documented. A separate description statement (within each 'enum' or 'bit' statement) SHOULD be present.

```
leaf foo {
  // INCORRECT
  type enumeration {
    enum one;
    enum two;
  }
  description
    "The foo enum...
     one: The first enum
     two: The second enum";
}

leaf foo {
  // CORRECT
  type enumeration {
    enum one {
      description "The first enum";
    }
    enum two {
      description "The second enum";
    }
  }
  description
    "The foo enum...  ";
}
```

4.11.4. Union Types

The YANG "union" type is evaluated by testing a value against each member type in the union. The first type definition that accepts a value as valid is the member type used. In general, member types SHOULD be ordered from most restrictive to least restrictive types.

In the following example, the "enumeration" type will never be matched because the preceding "string" type will match everything.

Incorrect:

```
type union {
  type string;
  type enumeration {
    enum up;
    enum down;
  }
}
```

Correct:

```
type union {
  type enumeration {
    enum up;
    enum down;
  }
  type string;
}
```

It is possible for different member types to match, depending on the input encoding format. In XML, all values are passed as string nodes, but in JSON there are different value types for numbers, booleans, and strings.

In the following example, a JSON numeric value will always be matched by the "int32" type but in XML the string value representing a number will be matched by the "string" type. The second version will match the "int32" member type no matter how the input is encoded.

Incorrect:

```
type union {
  type string;
  type int32;
}
```

Correct:

```
type union {
  type int32;
  type string;
}
```

4.11.5. Empty and Boolean

YANG provides an "empty" data type, which has one value (i.e., present). The default is "not present", which is not actually a value. When used within a list key, only one value can (and must) exist for this key leaf. The type "empty" SHOULD NOT be used for a key leaf since it is pointless.

There is really no difference between a leaf of type "empty" and a leaf-list of type "empty". Both are limited to one instance. The type "empty" SHOULD NOT be used for a leaf-list.

The advantage of using type "empty" instead of type "boolean" is that the default (not present) does not take up any bytes in a representation. The disadvantage is that the client may not be sure if an empty leaf is missing because it was filtered somehow or not

implemented. The client may not have a complete and accurate schema for the data returned by the server, and not be aware of the missing leaf.

The YANG "boolean" data type provides two values ("true" and "false"). When used within a list key, two entries can exist for this key leaf. Default values are ignored for key leafs, but a default statement is often used for plain boolean leafs. The advantage of the "boolean" type is that the leaf or leaf-list has a clear representation for both values. The default value is usually not returned unless explicitly requested by the client, so no bytes are used in a typical representation.

In general, the "boolean" data type SHOULD be used instead of the "empty" data type, as shown in the example below:

Incorrect:

```
leaf flag1 {
  type empty;
}
```

Correct:

```
leaf flag2 {
  type boolean;
  default false;
}
```

4.12. Reusable Type Definitions

If an appropriate derived type exists in any standard module, such as [RFC6991], then it SHOULD be used instead of defining a new derived type.

If an appropriate units identifier can be associated with the desired semantics, then a units statement SHOULD be present.

If an appropriate default value can be associated with the desired semantics, then a default statement SHOULD be present.

If a significant number of derived types are defined, and it is anticipated that these data types will be reused by multiple modules, then these derived types SHOULD be contained in a separate module or submodule, to allow easier reuse without unnecessary coupling.

The description statement MUST be present.

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

4.13. Reusable Groupings

A reusable grouping is a YANG grouping that can be imported by another module, and is intended for use by other modules. This is not the same as a grouping that is used within the module it is defined, but happens to be exportable to another module because it is defined at the top-level of the YANG module.

The following guidelines apply to reusable groupings, in order to make them as robust as possible:

- Clearly identify the purpose of the grouping in the "description" statement.
- There are 5 different XPath contexts in YANG (rpc/input, rpc/output, notification, config=true data nodes, and all data nodes). Clearly identify which XPath contexts are applicable or excluded for the grouping.
- Do not reference data outside the grouping in any "path", "must", or "when" statements.
- Do not include a "default" sub-statement on a leaf or choice unless the value applies on all possible contexts.
- Do not include a "config" sub-statement on a data node unless the value applies on all possible contexts.
- Clearly identify any external dependencies in the grouping "description" statement, such as nodes referenced by absolute path from a "path", "must", or "when" statement.

4.14. Data Definitions

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

- anyxml
- augment
- choice

- container
- extension
- feature
- grouping
- identity
- leaf
- leaf-list
- list
- notification
- rpc
- typedef

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

The 'anyxml' construct may be useful to represent an HTML banner containing markup elements, such as '' and '', and MAY be used in such cases. However, this construct SHOULD NOT be used if other YANG data node types can be used instead to represent the desired syntax and semantics.

It has been found that the 'anyxml' statement is not implemented consistently across all servers. It is possible that mixed mode XML will not be supported, or configuration anyxml nodes will not supported.

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

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

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

The "choice" statement is allowed to be directly present within a "case" statement in YANG 1.1. This needs to be considered carefully. Consider simply including the nested "choice" as additional "case" statements within the parent "choice" statement. Note that the "mandatory" and "default" statements within a nested "choice" statement only apply if the "case" containing the nested "choice" statement is first selected.

If a list defines any key leafs, then these leafs SHOULD be defined in order, as the first child nodes within the list. The key leafs MAY be in a different order in some cases, e.g., they are defined in a grouping, not inline in the list statement.

4.14.1. Non-Presence Containers

A non-presence container is used to organize data into specific subtrees. It is not intended to have semantics within the data model beyond this purpose, although YANG allows it (e.g., "must" statement within the non-presence container).

Example using container wrappers:

```
container top {
   container foos {
      list foo { ... }
   }
   container bars {
      list bar { ... }
   }
}
```

Example without container wrappers:

```
container top {
   list foo { ... }
   list bar { ... }
}
```

Use of non-presence containers to organize data is a subjective matter similar to use of sub-directories in a file system. Although these containers do not have any semantics, they can impact protocol operations for the descendant data nodes within a non-presence container, so use of these containers SHOULD be considered carefully.

The NETCONF and RESTCONF protocols do not currently support the ability to delete all list (or leaf-list) entries at once. This deficiency is sometimes avoided by use of a parent container (i.e., deleting the container also removes all child entries).

4.14.2. Top-Level Data Nodes

Use of top-level objects needs to be considered carefully:

- top-level siblings are not ordered
- top-level siblings not are not static, and depends on the modules that are loaded
- for sub-tree filtering, retrieval of a top-level leaf-list will be treated as a content-match node for all top-level-siblings
- a top-level list with many instances may impact performance

4.15. Operation Definitions

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

If the operation impacts system behavior in some way, it SHOULD be mentioned in the description statement.

If the operation is potentially harmful to system behavior in some way, it MUST be mentioned in the Security Considerations section of the document.

4.16. Notification Definitions

The description statement MUST be present.

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

If the notification refers to a specific resource instance, then this instance SHOULD be identified in the notification data. This is usually done by including 'leafref' leaf nodes with the key leaf values for the resource instance. For example:

```
  notification interface-up {
    description "Sent when an interface is activated.";
    leaf name {
      type leafref {
        path "/if:interfaces/if:interface/if:name";
      }
    }
  }
```

Note that there are no formal YANG statements to identify any data node resources associated with a notification. The description statement for the notification SHOULD specify if and how the notification identifies any data node resources associated with the specific event.

4.17. Feature Definitions

The YANG "feature" statement is used to define a label for a set of optional functionality within a module. The "if-feature" statement is used in the YANG statements associated with a feature. The description-stmt within a feature-stmt MUST specify any interactions with other features.

The set of YANG features defined in a module should be considered carefully. Very fine granular features increase interoperability complexity and should be avoided. A likely misuse of the feature mechanism is the tagging of individual leafs (e.g., counters) with separate features.

If there is a large set of objects associated with a YANG feature, then consider moving those objects to a separate module, instead of using a YANG feature. Note that the set of features within a module is easily discovered by the reader, but the set of related modules within the entire YANG library is not as easy to identity. Module names with a common prefix can help readers identity the set of related modules, but this assumes the reader will have discovered and installed all the relevant modules.

Another consideration for deciding whether to create a new module or add a YANG feature is the stability of the module in question. It may be desirable to have a stable base module that is not changed frequently. If new functionality is placed in a separate module, then the base module does not need to be republished. If it is designed as a YANG feature then the module will need to be republished.

If one feature requires implementation of another feature, then an "if-feature" statement SHOULD be used in the dependent "feature" statement.

For example, feature2 requires implementation of feature1:

```
feature feature1 {
  description "Some protocol feature";
}

feature feature2 {
  if-feature "feature1";
  description "Another protocol feature";
}
```

4.18. YANG Data Node Constraints

4.18.1. Controlling Quantity

The "min-elements" and "max-elements" statements can be use to control how many list or leaf-list instances are required for a particular data node. YANG constraint statements SHOULD be used to identify conditions that apply to all implementations of the data model. If platform-specific limitations (e.g., the "max-elements" supported for a particular list) are relevant to operations, then a data model definition statement (e.g., "max-ports" leaf) SHOULD be used to identify the limit.

4.18.2. must vs. when

The "must" and "when" YANG statements are used to provide cross-object referential tests. They have very different behavior. The "when" statement causes data node instances to be silently deleted as soon as the condition becomes false. A false "when" expression is not considered to be an error.

The "when" statement SHOULD be used together with the "augment" or "uses" statements to achieve conditional model composition. The condition SHOULD be based on static properties of the augmented entry (e.g., list key leafs).

The "must" statement causes a datastore validation error if the condition is false. This statement SHOULD be used for enforcing parameter value restrictions that involve more than one data node (e.g., end-time parameter must be after the start-time parameter).

4.19. Augment Statements

The YANG "augment" statement is used to define a set of data definition statements that will be added as child nodes of a target data node. The module namespace for these data nodes will be the augmenting module, not the augmented module.

A top-level "augment" statement SHOULD NOT be used if the target data

node is in the same module or submodule as the evaluated "augment" statement. The data definition statements SHOULD be added inline instead.

4.19.1. Conditional Augment Statements

The "augment" statement is often used together with the "when" statement and/or "if-feature" statement to make the augmentation conditional on some portion of the data model.

The following example from [RFC7223] shows how a conditional container called "ethernet" is added to the "interface" list only for entries of the type "ethernetCsmacd".

```
augment "/if:interfaces/if:interface" {
    when "if:type = 'ianaift:ethernetCsmacd'";

    container ethernet {
        leaf duplex {
            ...
        }
    }
}
```

4.19.2. Conditionally Mandatory Data Definition Statements

YANG has very specific rules about how configuration data can be updated in new releases of a module. These rules allow an "old client" to continue interoperating with a "new server".

If data nodes are added to an existing entry, the old client MUST NOT be required to provide any mandatory parameters that were not in the original module definition.

It is possible to add conditional augment statements such that the old client would not know about the new condition, and would not specify the new condition. The conditional augment statement can contain mandatory objects only if the condition is false unless explicitly requested by the client.

Only a conditional augment statement that uses the "when" statement form of condition can be used in this manner. The YANG features enabled on the server cannot be controlled by the client in any way, so it is not safe to add mandatory augmenting data nodes based on the "if-feature" statement.

The XPath "when" statement condition MUST NOT reference data outside of target data node because the client does not have any control over

this external data.

In the following dummy example, it is OK to augment the "interface" entry with "mandatory-leaf" because the augmentation depends on support for "some-new-iftype". The old client does not know about this type so it would never select this type, and therefore not be adding a mandatory data node.

```
  module example-module {
    namespace "tag:example.com,2017:example-module";
    prefix mymod;

    import iana-if-type { prefix iana; }
    import ietf-interfaces { prefix if; }

    identity some-new-iftype {
       base iana:iana-interface-type;
    }

    augment "/if:interfaces/if:interface" {
       when "if:type = 'mymod:some-new-iftype'";

       leaf mandatory-leaf {
          mandatory true;
          ...
       }
    }
  }
```

Note that this practice is safe only for creating data resources. It is not safe for replacing or modifying resources if the client does not know about the new condition. The YANG data model MUST be packaged in a way that requires the client to be aware of the mandatory data nodes if it is aware of the condition for this data. In the example above, the "some-new-iftype" identity is defined in the same module as the "mandatory-leaf" data definition statement.

This practice is not safe for identities defined in a common module such as "iana-if-type" because the client is not required to know about "my-module" just because it knows about the "iana-if-type" module.

4.20. Deviation Statements

Per RFC 7950, 7.20.3, the YANG "deviation" statement is not allowed to appear in IETF YANG modules, but it can be useful for documenting server capabilities. Deviation statements are not reusable and typically not shared across all platforms.

There are several reasons that deviations might be needed in an implementation, e.g., an object cannot be supported on all platforms, or feature delivery is done in multiple development phases. Deviation statements can also be used to add annotations to a module, which does not affect the conformance requirements for the module.

It is suggested that deviation statements be defined in separate modules from regular YANG definitions. This allows the deviations to be platform-specific and/or temporary.

The order that deviation statements are evaluated can affect the result. Therefore multiple deviation statements in the same module, for the same target object, SHOULD NOT be used.

The "max-elements" statement is intended to describe an architectural limit to the number of list entries. It is not intended to describe platform limitations. It is better to use a "deviation" statement for the platforms that have a hard resource limit.

Example documenting platform resource limits:

Wrong: (max-elements in the list itself)

```
container backups {
  list backup {
     ...
     max-elements  10;
     ...
  }
}
```

Correct: (max-elements in a deviation)

```
deviation /bk:backups/bk:backup {
  deviate add {
     max-elements  10;
  }
}
```

4.21. Extension Statements

The YANG "extension" statement is used to specify external definitions. This appears in the YANG syntax as an "unknown-statement". Usage of extension statements in a published module needs to be considered carefully.

The following guidelines apply to the usage of YANG extensions:

- The semantics of the extension MUST NOT contradict any YANG statements. Extensions can add semantics not covered by the normal YANG statements.

- The module containing the extension statement MUST clearly identify the conformance requirements for the extension. It should be clear whether all implementations of the YANG module containing the extension need to also implement the extension. If not, identify what conditions apply that would require implementation of the extension.

- The extension MUST clearly identify where it can be used within other YANG statements.

- The extension MUST clearly identify if YANG statements or other extensions are allowed or required within the extension as sub-statements.

4.22. Data Correlation

Data can be correlated in various ways, using common data types, common data naming, and common data organization. There are several ways to extend the functionality of a module, based on the degree of coupling between the old and new functionality:

- inline: update the module with new protocol-accessible objects. The naming and data organization of the original objects is used. The new objects are in the original module namespace.

- augment: create a new module with new protocol-accessible objects that augment the original data structure. The naming and data organization of the original objects is used. The new objects are in the new module namespace.

- mirror: create new objects in a new module or the original module, except use new a naming scheme and data location. The naming can be coupled in different ways. Tight coupling is achieved with a "leafref" data type, with the "require-instance" sub-statement set to "true". This method SHOULD be used.

If the new data instances are not limited to the values in use in the original data structure, then the "require-instance" sub-statement MUST be set to "false". Loose coupling is achieved by using key leafs with the same data type as the original data structure. This has the same semantics as setting the "require-instance" sub-statement to "false".

The relationship between configuration and operational state has been

clarified in NMDA [I-D.ietf-netmod-revised-datastores].

4.22.1. Use of Leafref for Key Correlation

Sometimes it is not practical to augment a data structure. For example, the correlated data could have different keys or contain mandatory nodes.

The following example shows the use of the "leafref" data type for data correlation purposes:

Not preferred:

```
list foo {
   key name;
   leaf name {
     type string;
   }
   ...
}

list foo-addon {
   key name;
   config false;
   leaf name {
     type string;
   }
   ...
}
```

Preferred:

```
      list foo {
         key name;
         leaf name {
           type string;
         }
         ...
      }

      list foo-addon {
         key name;
         config false;
         leaf name {
           type leafref {
             path "/foo/name";
             require-instance false;
           }
         }
         leaf addon {
           type string;
           mandatory true;
         }
      }
```

4.23. Operational State

The modeling of operational state with YANG has been refined over time. At first, only data that has a "config" statement value of "false" was considered to be operational state. This data was not considered to be part of any datastore, which made YANG XPath definition much more complicated.

Operational state is now modeled using YANG according to the new NMDA [I-D.ietf-netmod-revised-datastores], and is now conceptually contained in the operational state datastore, which also includes the operational values of configuration data. There is no longer any need to duplicate data structures to provide separate configuration and operational state sections.

This section describes some data modeling issues related to operational state, and guidelines for transitioning YANG data model design to be NMDA-compatible.

4.23.1. Combining Operational State and Configuration Data

If possible, operational state SHOULD be combined with its associated configuration data. This prevents duplication of key leafs and ancestor nodes. It also prevents race conditions for retrieval of dynamic entries, and allows configuration and operational state to be

retrieved together with minimal message overhead.

```
container foo {
  ...
  // contains config=true and config=false nodes that have
  // no corresponding config=true object (e.g., counters)
}
```

4.23.2. Representing Operational Values of Configuration Data

If possible the same data type SHOULD be used to represent the configured value and the operational value, for a given leaf or leaf-list object.

Sometimes the configured value set is different than the operational value set for that object. For example, the "admin-state" and "oper-state" leafs in [RFC7223]. In this case a separate object MAY be used to represent the configured and operational values.

Sometimes the list keys are not identical for configuration data and the corresponding operational state. In this case separate lists MAY be used to represent the configured and operational values.

If it is not possible to combine configuration and operational state, then the keys used to represent list entries SHOULD be the same type. The "leafref" data type SHOULD be used in operational state for key leafs that have corresponding configuration instances. The "require-instance" statement MAY be set to "false" (in YANG 1.1 modules only) to indicate instances are allowed in the operational state that do not exist in the associated configuration data.

The need to replicate objects or define different operational state objects depends on the data model. It is not possible to define one approach that will be optimal for all data models.

Designers SHOULD describe and justify any NMDA exceptions in detail, such as the use of separate subtrees and/or separate leafs. The "description" statements for both the configuration and the operational state SHOULD be used for this purpose.

4.23.3. NMDA Transition Guidelines

YANG modules SHOULD be designed assuming they will be used on servers supporting the operational state datastore. With this in mind, YANG modules SHOULD define config "false" wherever they make sense to the data model. Config "false" nodes SHOULD NOT be defined to provide the operational value for configuration nodes, except when the value space of a configured and operational values may differ, in which

case a distinct config "false" node SHOULD be defined to hold the operational value for the configured node.

The following guidelines are meant to help modelers develop YANG modules that will maximize the utility of the model with both current and new implementations.

New modules and modules that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible, as described in Section 4.23.1. This transition MAY be deferred if the module does not contain any configuration datastore objects.

The remaining are options that MAY be followed during the time that NMDA mechanisms are being defined.

(a) Modules that require immediate support for the NMDA features SHOULD be structured for NMDA. A temporary non-NMDA version of this type of module MAY exist, either an existing model or a model created either by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA module will allow temporary bridging of the time period until NMDA implementations are available.

(b) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] has been restructured as an NMDA-compatible model in [I-D.ietf-netmod-rfc7223bis]. The "/interfaces-state" hierarchy has been marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

(c) For models that augment models which have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA-compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

4.23.3.1. Temporary non-NMDA Modules

A temporary non-NMDA module allows a non-NMDA aware client to access operational state from an NMDA-compliant server. It contains the top-level config=false data nodes that would have been defined in a legacy YANG module (before NMDA).

A server that needs to support both NMDA and non-NMDA clients can advertise both the new NMDA module and the temporary non-NMDA module. A non-NMDA client can use separate "foo" and "foo-state" subtrees, except the "foo-state" subtree is located in a different (temporary) module. The NMDA module can be used by a non-NMDA client to access the conventional configuration datastores, and the deprecated <get> operation to access nested config=false data nodes.

To create the temporary non-NMDA model from an NMDA model, the following steps can be taken:

- Change the module name by appending "-state" to the original module name
- Change the namespace by appending "-state" to the original namespace value
- Change the prefix by appending "-s" to the original prefix value
- Add an import to the original module (e.g., for typedef definitions)
- Retain or create only the top-level nodes that have a "config" statement value "false". These subtrees represent config=false data nodes that were combined into the configuration subtree, and therefore not available to non-NMDA aware clients. Set the "status" statement to "deprecated" for each new node.
- The module description SHOULD clearly identify the module as a temporary non-NMDA module

4.23.3.2. Example: Create a New NMDA Module

Create an NMDA-compliant module, using combined configuration and state subtrees, whenever possible.

```
    module example-foo {
      namespace "urn:example.com:params:xml:ns:yang:example-foo";
      prefix "foo";

      container foo {
        // configuration data child nodes
        // operational value in operational state datastore only
        // may contain config=false nodes as needed
      }
    }
```

4.23.3.3. Example: Convert an old Non-NMDA Module

Do not remove non-compliant objects from existing modules. Instead, change the status to "deprecated". At some point, usually after 1 year, the status MAY be changed to "obsolete".

Old Module:

```
    module example-foo {
      namespace "urn:example.com:params:xml:ns:yang:example-foo";
      prefix "foo";

      container foo {
        // configuration data child nodes
      }

      container foo-state {
        config false;
        // operational state child nodes
      }
    }
```

Converted NMDA Module:

```
     module example-foo {
       namespace "urn:example.com:params:xml:ns:yang:example-foo";
       prefix "foo";

       container foo {
         // configuration data child nodes
         // operational value in operational state datastore only
         // may contain config=false nodes as needed
         // will contain any data nodes from old foo-state
       }

       // keep original foo-state but change status to deprecated
       container foo-state {
         config false;
         status deprecated;
         // operational state child nodes
       }
    }
```

4.23.3.4. Example: Create a Temporary NMDA Module:

Create a new module that contains the top-level operational state data nodes that would have been available before they were combined with configuration data nodes (to be NMDA compliant).

```
     module example-foo-state {
       namespace "urn:example.com:params:xml:ns:yang:example-foo-state";
       prefix "foo-s";

       // import new or converted module; not used in this example
       import example-foo { prefix foo; }

       container foo-state {
         config false;
         status deprecated;
         // operational state child nodes
       }
    }
```

4.24. Performance Considerations

It is generally likely that certain YANG statements require more runtime resources than other statements. Although there are no performance requirements for YANG validation, the following information MAY be considered when designing YANG data models:

- Lists are generally more expensive than containers
- "when-stmt" evaluation is generally more expensive than "if-feature" or "choice" statements
- "must" statement is generally more expensive than "min-entries", "max-entries", "mandatory", or "unique" statements
- "identityref" leafs are generally more expensive than "enumeration" leafs
- "leafref" and "instance-identifier" types with "require-instance" set to true are generally more expensive than if "require-instance" is set to false

4.25. Open Systems Considerations

Only the modules imported by a particular module can be assumed to be present in an implementation. An open system MAY include any combination of YANG modules.

4.26. Guidelines for YANG 1.1 Specific Constructs

The set of YANG 1.1 guidelines will grow as operational experience is gained with the new language features. This section contains an initial set of guidelines for new YANG 1.1 language features.

4.26.1. Importing Multiple Revisions

Standard modules SHOULD NOT import multiple revisions of the same module into a module. This MAY be done if independent definitions (e.g. enumeration typedefs) from specific revisions are needed in the importing module.

4.26.2. Using Feature Logic

The YANG 1.1 feature logic is much more expressive than YANG 1.0. A "description" statement SHOULD describe the "if-feature" logic in text, to help readers understand the module.

YANG features SHOULD be used instead of the "when" statement, if possible. Features are advertised by the server and objects conditional by if-feature are conceptually grouped together. There is no such commonality supported for "when" statements.

Features generally require less server implementation complexity and runtime resources than objects that use "when" statements. Features are generally static (i.e., set when module is loaded and not changed

at runtime). However every client edit might cause a "when" statement result to change.

4.26.3. anyxml vs. anydata

The "anyxml" statement MUST NOT be used to represent a conceptual subtree of YANG data nodes. The "anydata" statement MUST be used for this purpose.

4.26.4. action vs. rpc

The use of "action" statements or "rpc" statements is a subjective design decision. RPC operations are not associated with any particular data node. Actions are associated with a specific data node definition. An "action" statement SHOULD be used if the protocol operation is specific to a subset of all data nodes instead of all possible data nodes.

The same action name MAY be used in different definitions within different data node. For example, a "reset" action defined with a data node definition for an interface might have different parameters than for a power supply or a VLAN. The same action name SHOULD be used to represent similar semantics.

The NETCONF Access Control Model (NACM) [I-D.ietf-netconf-rfc6536bis] does not support parameter-based access control for RPC operations. The user is given permission (or not) to invoke the RPC operation with any parameters. For example, if each client is only allowed to reset their own interface, then NACM cannot be used.

For example, NACM cannot enforce access access control based on the value of the "interface" parameter, only the "reset" operation itself:

```
rpc reset {
  input {
    leaf interface {
      type if:interface-ref;
      mandatory true;
      description "The interface to reset.";
    }
  }
}
```

However, NACM can enforce access access control for individual interface instances, using a "reset" action, If the user does not have read access to the specific "interface" instance, then it cannot invoke the "reset" action for that interface instance:

```
container interfaces {
  list interface {
    ...
    action reset { }
  }
}
```

4.27. Updating YANG Modules (Published vs. Unpublished)

YANG modules can change over time. Typically, new data model definitions are needed to support new features. YANG update rules defined in section 11 of [RFC7950] MUST be followed for published modules. They MAY be followed for unpublished modules.

The YANG update rules only apply to published module revisions. Each organization will have their own way to identify published work which is considered to be stable, and unpublished work which is considered to be unstable. For example, in the IETF, the RFC document is used for published work, and the Internet-Draft is used for unpublished work.

5. IANA Considerations

-- RFC Ed: These registries need to be updated to reference this RFC instead of RFC 6087 for the ietf-template module, and remove this note.

This document registers one URI in the IETF XML registry [RFC3688].

The following registration has been made in [RFC6087] and updated by this document.

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

Registrant Contact: The IESG.

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

The following assignment has been made in [RFC6087] and updated by this document in the YANG Module Names Registry, or the YANG module template in Appendix C.

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

YANG Registry Assignment

6. Security Considerations

This document defines documentation guidelines for NETCONF or RESTCONF content defined with the YANG data modeling language, and therefore does not introduce any new or increased security risks into the management system.

7. Acknowledgments

The structure and contents of this document are adapted from [RFC4181], guidelines for MIB Documents, by C. M. Heard.

The working group thanks Martin Bjorklund, Juergen Schoenwaelder, Ladislav Lhotka, Jernej Tuljak, and Lou Berger for their extensive reviews and contributions to this document.

8. References

8.1. Normative References

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

[ID-Guidelines]
Housley, R., Ed., "Guidelines to Authors of Internet-Drafts", December 2010, <https://www.ietf.org/standards/ids/guidelines/>.

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

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[W3C.REC-xpath-19991116]
Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium

Recommendation REC-xpath-19991116, November 1999,
<http://www.w3.org/TR/1999/REC-xpath-19991116>.

8.2. Informative References

[I-D.flanagan-7322bis]
Flanagan, H. and R. Editor, "RFC Style Guide",
draft-flanagan-7322bis-02 (work in progress),
September 2017.

[I-D.ietf-netconf-rfc6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration
Access Control Module", draft-ietf-netconf-rfc6536bis-09
(work in progress), December 2017.

[I-D.ietf-netmod-rfc7223bis]
Bjorklund, M., "A YANG Data Model for Interface
Management", draft-ietf-netmod-rfc7223bis-03 (work in
progress), January 2018.

[I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for
Routing Management (NMDA Version)",
draft-ietf-netmod-rfc8022bis-11 (work in progress),
January 2018.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams",
draft-ietf-netmod-yang-tree-diagrams-06 (work in
progress), February 2018.

[RFC-STYLE]
Braden, R., Ginoza, S., and A. Hagens, "RFC Document
Style", September 2009,
<http://www.rfc-editor.org/styleguide/>.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision
3", BCP 9, RFC 2026, DOI 10.17487/RFC2026, October 1996,
<http://www.rfc-editor.org/info/rfc2026>.

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme",
RFC 4151, DOI 10.17487/RFC4151, October 2005,
<http://www.rfc-editor.org/info/rfc4151>.

[RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB
Documents", BCP 111, RFC 4181, September 2005.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG

Data Model Documents", RFC 6087, January 2011.

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

[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, March 2012.

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

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

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <http://www.rfc-editor.org/info/rfc7322>.

[RFC7841] Halpern, J., Ed., Daigle, L., Ed., and O. Kolkman, Ed., "RFC Streams, Headers, and Boilerplates", RFC 7841, DOI 10.17487/RFC7841, May 2016, <http://www.rfc-editor.org/info/rfc7841>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Appendix A. Change Log

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

A.1. v19 to v20

- fix examples

A.2. v18 to v19

- address IESG ballot comments

A.3. v17 to v18

- address Area Director review comments Part 2
- clarify preferred list key order

A.4. v16 to v17

- address Area Director review comments Part 1

A.5. v15 to v16

- address Area Director review comments posted 2018-01-25

A.6. v15 to v16

- address document shephard comments posted 2018-01-15
- add yang-version to template module

A.7. v14 to v15

- changed Intended status from Informational to BCP
- update tree diagram guidelines section
- Change IANA template to list IESG instead of NETMOD WG as the Registrant
- Update some references

A.8. v13 to v14

- Replaced sec. 4.23 Operational Data with Operational Data from NMDA text by Lou Berger and Kent Watsen

- Added NMDA Terms section
- Changed term operational data to operational state
- Clarified that reference-stmt SHOULD be present in import-stmt

A.9. v12 to v13

- Clarify that the revision-date SHOULD be used in a CODE BEGINS YANG file extraction macro.
- Clarify the IANA requirements section wrt/ XML namespace and YANG module name registries.
- Clarify YANG Usage section wrt/ XML and/or JSON encoding format.
- Update Operation Data section to consider revised datastores.
- Add reference to YANG Tree Diagrams and update 2 sections that use this reference.
- Add reference to Revised Datastores and guidelines drafts

A.10. v11 to v12

- fix incorrect location of new Module Usage Examples section

A.11. v10 to v11

- updated YANG tree diagram syntax to align with pyang 1.7.1
- added general guideline to include module usage examples

A.12. v09 to v10

- clarified <CODE BEGINS> is only for normative modules
- clarified example module namespace URI conventions
- clarified pyang usage for normative and example modules
- updated YANG tree diagrams section with text from RFC 8022

A.13. v08 to v09

- fixed references

- added mention of RESTCONF to abstract and intro
- created separate section for code components
- fixed document status

A.14. v07 to v08

- changed CODE BEGINS guideline for example modules
- updated tree diagram guidelines
- clarified published and unpublished terms
- added section on Empty and Boolean data types
- clarified how to update the revision statement
- updated operational state guidelines
- added 'YANG fragment' to terminology section

A.15. v06 to v07

- update contact statement guideline
- update example modules guidelines
- add guidelines on top-level data nodes
- add guideline on use of NP containers
- added guidelines on union types
- add guideline on deviations
- added section on open systems considerations
- added guideline about definitions reserved for future use

A.16. v05 to v06

- Changed example 'my-module' to 'example-module'
- Added section Updating YANG Modules (Published vs. Unpublished)
- Added Example Modules section

- Added "<EXAMPLE BEGINS>" convention for full example modules
- Added section on using action vs. rpc
- Changed term "operational state" to "operational data"
- Added section on YANG Data Node Constraints
- Added guidelines on using must vs. when statements
- Made ietf-foo module validate for I-D submission

A.17. v04 to v05

- Clarified that YANG 1.1 SHOULD be used but YANG 1.0 MAY be used if no YANG 1.1 features needed
- Changed SHOULD follow YANG naming conventions to MUST follow (for standards track documents only)
- Clarified module naming conventions for normative modules, example modules, and modules from other SDOs.
- Added prefix value selection guidelines
- Added new section on guidelines for reusable groupings
- Made header guidelines less IETF-specific
- Added new section on guidelines for extension statements
- Added guidelines for nested "choice" statement within a "case" statement

A.18. v03 ot v04

- Added sections for deviation statements and performance considerations
- Added YANG 1.1 section
- Updated YANG reference from 1.0 to 1.1

A.19. v02 to v03

- Updated draft based on github data tracker issues added by Benoit Clause (Issues 12 - 18)

A.20. v01 to v02

- Updated draft based on mailing list comments.

A.21. v00 to v01

All issues from the issue tracker have been addressed.

https://github.com/netmod-wg/rfc6087bis/issues

- Issue 1: Tree Diagrams: Added 'tree-diagrams' section so RFCs with YANG modules can use an Informative reference to this RFC for tree diagrams. Updated guidelines to reference this RFC when tree diagrams are used

- Issue 2: XPath function restrictions: Added paragraphs in XPath usage section for 'id', 'namespace-uri', 'name', and 'lang' functions

- Issue 3: XPath function document order issues: Added paragraph in XPath usage section about node-set ordering for 'local-name', 'namespace-uri', 'name', 'string' and 'number' functions. Also any function that implicitly converts a node-set to a string.

- Issue 4: XPath preceding-sibling and following-sibling: Checked and text in XPath usage section already has proposed text from Lada.

- Issue 5: XPath 'when-stmt' reference to descendant nodes: Added exception and example in XPath Usage section for augmented nodes.

- Issue 6: XPath numeric conversions: Changed 'numeric expressions' to 'numeric and boolean expressions'

- Issue 7: XPath module containment: Added sub-section on XPath wildcards

- Issue 8: status-stmt usage: Added text to Lifecycle Management section about transitioning from active to deprecated and then to obsolete.

- Issue 9: resource identification in notifications: Add text to Notifications section about identifying resources and using the leafref data type.

- Issue 10: single quoted strings: Added text to Data Types section about using a single-quoted string for patterns.

Appendix B. Module Review Checklist

This section is adapted from RFC 4181.

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

- I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see https://www.ietf.org/id-info/guidelines.html), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.

- Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in https://www.ietf.org/id-info/guidelines.html.

- Copyright Notice -- verify that the draft has the appropriate text regarding the rights that document contributers provide to the IETF Trust [RFC5378]. Verify that it contains the full IETF Trust copyright notice at the beginning of the document. The IETF Trust Legal Provisions (TLP) can be found at:

  https://trustee.ietf.org/license-info/

- Security Considerations section -- verify that the draft uses the latest approved template from the OPS area website (https://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines) and that the guidelines therein have been followed.

- IANA Considerations section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

  XML Namespace Registry: Register the YANG module namespace.

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

- References -- verify that the references are properly divided between normative and informative references, that RFC 2119 and RFC 8174 are included as normative references if the terminology defined therein is used in the document, that all references required by the boilerplate are present, that all YANG modules containing imported items are cited as normative references, and that all citations point to the most current RFCs unless there is a valid reason to do otherwise (for example, it is OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module). If a YANG module contains reference or description statements that refer to an Internet Draft (I-D), then the I-D is included as an Informative Reference.

- License -- verify that the draft contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in Section 3.1. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

  https://trustee.ietf.org/license-info/

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

- Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

  https://trac.tools.ietf.org/wg/netconf/trac/wiki

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

Appendix C. YANG Module Template

```
   <CODE BEGINS> file "ietf-template@2016-03-20.yang"

      module ietf-template {

        yang-version 1.1;

        // replace this string with a unique namespace URN value
        namespace
          "urn:ietf:params:xml:ns:yang:ietf-template";

        // replace this string, and try to pick a unique prefix
        prefix "temp";

        // import statements here: e.g.,
        // import ietf-yang-types { prefix yang; }
        // import ietf-inet-types { prefix inet; }

        // identify the IETF working group if applicable
        organization
           "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

        // update this contact statement with your info
        contact
           "WG Web:   <http://tools.ietf.org/wg/your-wg-name/>
            WG List:  <mailto:your-wg-name@ietf.org>

            Editor:   your-name
                      <mailto:your-email@example.com>";


        // replace the first sentence in this description statement.
        // replace the copyright notice with the most recent
        // version, if it has been updated since the publication
        // of this document
        description
         "This module defines a template for other YANG modules.

          Copyright (c) <insert year> IETF Trust and the persons
          identified as authors of the code.  All rights reserved.

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

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

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

        reference "RFC XXXX: <Replace With Document Title>";


        // RFC Ed.: remove this note
        // Note: extracted from RFC XXXX


        // replace '2016-03-20' with the module publication date
        // The format is (year-month-day)
        revision "2016-03-20" {
          description "what changed in this revision";
          reference "document containing this module";
        }

        // extension statements

        // feature statements

        // identity statements

        // typedef statements

        // grouping statements

        // data definition statements

        // augment statements

        // rpc statements

        // notification statements

        // DO NOT put deviation statements in a published module

      }

   <CODE ENDS>
```

Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems

November 03, 2016

A YANG Data Model for Routing Management
draft-ietf-netmod-routing-cfg-25

Abstract

This document contains a specification of three YANG modules and one submodule. 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 control plane protocols, route filters and other functions. The core routing data model provides common building blocks for such extensions -- routes, routing information bases (RIB), and control plane 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 May 7, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document contains a specification of the following YANG modules:

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

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

- Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast. Its submodule "ietf-ipv6-router-advertisements" also augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules 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 control plane protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- client,
- message,
- protocol operation,
- server.

The following terms are defined in [RFC7950]:

- action,
- augment,
- configuration data,
- container,
- container with presence,
- data model,
- data node,
- feature,
- leaf,
- list,
- mandatory node,
- module,
- schema tree,
- state data,
- 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 5.2 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.

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

2.3. Prefixes in Data Node Names

In this document, names of data nodes, actions and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC7223]
ip	ietf-ip	[RFC7277]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

3. Objectives

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

- The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).

- 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.

- On the other hand, the core routing framework must allow for complicated implementations involving multiple routing information bases (RIB) and multiple control plane protocols, as well as controlled redistributions of routing information.

- Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules and one submodule. 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. Module "ietf-ipv6-unicast-routing" has a submodule, "ietf-ipv6-router-advertisements", that augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with configuration variables for IPv6 router advertisements as required by [RFC4861]. 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 router-id?
   +--rw control-plane-protocols
   |  +--rw control-plane-protocol* [type name]
   |     +--rw type
   |     +--rw name
   |     +--rw description?
   |     +--rw static-routes
   |        +--rw v6ur:ipv6
   |        |     ...
   |        +--rw v4ur:ipv4
   |              ...
   +--rw ribs
      +--rw rib* [name]
         +--rw name
         +--rw address-family?
         +--rw description?
```

Figure 1: Configuration data hierarchy.

```
+--ro routing-state
   +--ro router-id?
   +--ro interfaces
   |  +--ro interface*
   +--ro control-plane-protocols
   |  +--ro control-plane-protocol* [type name]
   |     +--ro type
   |     +--ro name
   +--ro ribs
      +--ro rib* [name]
         +--ro name
         +--ro address-family
         +--ro default-rib?
         +--ro routes
         |  +--ro route*
         |        ...
```

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: routes, RIBs containing lists of routes, and control plane 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, such as "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.

An example can be seen in Appendix D: the "/routing-state/ribs/rib" list has two system-controlled entries named "ipv4-master" and "ipv6-master".

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.

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. Route

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

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

- "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among routes with the same destination prefix. A lower value means a more preferred route.

- "next-hop": determines the outgoing interface and/or next-hop address(es), other operation to be performed with a packet.

Routes are primarily state data that appear as entries of RIBs (Section 5.2) 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 attributes defined for RIB routes.

5.2. Routing Information Base (RIB)

Every implementation of the core routing data model 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/ribs/rib". The contents of RIBs are controlled and manipulated by control plane 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 5.3.1.

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB. Its role is explained in Section 5.3.

Simple router implementations that do not advertise the feature "multiple-ribs" will typically create one system-controlled RIB per 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.

The following action (see Section 7.15 of [RFC7950]) is defined for the "rib" list:

- active-route -- return the active RIB route for the destination address that is specified as the action’s input parameter.

5.3. Control Plane Protocol

The core routing data model provides an open-ended framework for defining multiple control plane protocol instances, e.g., for Layer 3 routing protocols. Each control plane protocol instance MUST be assigned a type, which is an identity derived from the "rt:control-plane-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.3.1).

Multiple control plane protocol instances of the same type MAY be configured.

5.3.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types -- "direct" and "static". Both are in fact pseudo-protocols, which means that they are confined to the local device and do not exchange any routing information with adjacent routers.

Every implementation of the core routing data model MUST provide 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.

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.

5.3.2. Defining New Control Plane Protocols

It is expected that future YANG modules will create data models for additional control plane protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to integrate it into the core routing framework in the following way:

- A new identity MUST be defined for the control plane protocol and its base identity MUST be set to "rt:control-plane-protocol", or to an identity derived from "rt:control-plane-protocol".

- 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:routing-state/rt:ribs/rt:rib/rt:output/rt:route,

  and possibly other places in the configuration, state data, notifications, and input/output parameters of actions or RPC operations.

- Configuration parameters and/or state data for the new protocol can be defined by augmenting the "control-plane-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 (or derived from) 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.

5.4. Parameters of IPv6 Router Advertisements

YANG module "ietf-ipv6-router-advertisements" (Section 9.1), which is a submodule of the "ietf-ipv6-unicast-routing" module, augments the configuration and state data of IPv6 interfaces with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

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

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

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

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-router-advertisements" submodule therefore stipulates the former behavior with constant values.

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@2016-11-03.yang"

```
module ietf-routing {

  yang-version "1.1";

  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:   <https://tools.ietf.org/wg/netmod/>
```

```
        WG List:  <mailto:netmod@ietf.org>

        WG Chair: Lou Berger
                  <mailto:lberger@labn.net>

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

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>

        Editor:   Acee Lindem
                  <mailto:acee@cisco.com>";

    description
      "This YANG module defines essential components for the management
       of a routing subsystem.

       Copyright (c) 2016 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
       (https://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 (https://tools.ietf.org/html/rfc2119).

       This version of this YANG module is part of RFC XXXX
       (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
       full legal notices.";

    revision 2016-11-03 {
      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 supported address family
         and make them also the default RIBs. These RIBs then appear as
         entries of the list /routing-state/ribs/rib.";
    }

    feature router-id {
      description
        "This feature indicates that the server supports configuration
         of an explicit 32-bit router ID that is used by some routing
         protocols.

         Servers that do not advertise this feature set a router ID
         algorithmically, usually to one of configured IPv4 addresses.
         However, this algorithm is implementation-specific.";
    }

    /* Identities */

    identity address-family {
      description
        "Base identity from which identities describing address
         families are derived.";
    }

    identity ipv4 {
      base address-family;
      description
        "This identity represents IPv4 address family.";
    }

    identity ipv6 {
      base address-family;
      description
        "This identity represents IPv6 address family.";
    }

    identity control-plane-protocol {
      description
        "Base identity from which control plane protocol identities are
         derived.";
    }

    identity routing-protocol {
      base control-plane-protocol;
```

```
    description
      "Identity from which Layer 3 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 route-preference {
    type uint32;
    description
      "This type is used for route preferences.";
  }

  /* Groupings */

  grouping address-family {
    description
      "This grouping provides a leaf identifying an address
       family.";
    leaf address-family {
      type identityref {
        base address-family;
      }
      mandatory "true";
      description
        "Address family.";
    }
  }

  grouping router-id {
    description
      "This grouping provides router ID.";
    leaf router-id {
      type yang:dotted-quad;
      description
```

```
          "A 32-bit number in the form of a dotted quad that is used by
           some routing protocols identifying a router.";
        reference
          "RFC 2328: OSPF Version 2.";
      }
    }

    grouping special-next-hop {
      description
        "This grouping provides a leaf with an enumeration of special
         next-hops.";
      leaf special-next-hop {
        type enumeration {
          enum blackhole {
            description
              "Silently discard the packet.";
          }
          enum unreachable {
            description
              "Discard the packet and notify the sender with an error
               message indicating that the destination host is
               unreachable.";
          }
          enum prohibit {
            description
              "Discard the packet and notify the sender with an error
               message indicating that the communication is
               administratively prohibited.";
          }
          enum receive {
            description
              "The packet will be received by the local system.";
          }
        }
        description
          "Special next-hop options.";
      }
    }

    grouping next-hop-content {
      description
        "Generic parameters of next-hops in static routes.";
      choice next-hop-options {
        mandatory "true";
        description
          "Options for next-hops in static routes.

           It is expected that further cases will be added through
```

```
           augments from other modules.";
        case simple-next-hop {
          description
            "This case represents a simple next hop consisting of the
             next-hop address and/or outgoing interface.

             Modules for address families MUST augment this case with a
             leaf containing a next-hop address of that address
             family.";
          leaf outgoing-interface {
            type if:interface-ref;
            description
              "Name of the outgoing interface.";
          }
        }
        case special-next-hop {
          uses special-next-hop;
        }
        case next-hop-list {
          container next-hop-list {
            description
              "Container for multiple next-hops.";
            list next-hop {
              key "index";
              description
                "An entry of a next-hop list.

                 Modules for address families MUST augment this list
                 with a leaf containing a next-hop address of that
                 address family.";
              leaf index {
                type string;
                description
                  "An user-specified identifier utilised to uniquely
                   reference the next-hop entry in the next-hop list.
                   The value of this index has no semantic meaning
                   other than for referencing the entry.";
              }
              leaf outgoing-interface {
                type if:interface-ref;
                description
                  "Name of the outgoing interface.";
              }
            }
          }
        }
      }
    }
```

```
     grouping next-hop-state-content {
       description
         "Generic parameters of next-hops in state data.";
       choice next-hop-options {
         mandatory "true";
         description
           "Options for next-hops in state data.

            It is expected that further cases will be added through
            augments from other modules, e.g., for recursive
            next-hops.";
         case simple-next-hop {
           description
             "This case represents a simple next hop consisting of the
              next-hop address and/or outgoing interface.

              Modules for address families MUST augment this case with a
              leaf containing a next-hop address of that address
              family.";
           leaf outgoing-interface {
             type if:interface-state-ref;
             description
               "Name of the outgoing interface.";
           }
         }
         case special-next-hop {
           uses special-next-hop;
         }
         case next-hop-list {
           container next-hop-list {
             description
               "Container for multiple next-hops.";
             list next-hop {
               description
                 "An entry of a next-hop list.

                  Modules for address families MUST augment this list
                  with a leaf containing a next-hop address of that
                  address family.";
               leaf outgoing-interface {
                 type if:interface-state-ref;
                 description
                   "Name of the outgoing interface.";
               }
             }
           }
         }
       }
```

```
  }

  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 */

  container routing-state {
    config "false";
    description
      "State data of the routing subsystem.";
    uses router-id {
      description
        "Global router ID.

         It may be either configured or assigned algorithmically by
         the implementation.";
    }
    container interfaces {
      description
        "Network layer interfaces used for routing.";
      leaf-list interface {
        type if:interface-state-ref;
```

```
          description
            "Each entry is a reference to the name of a configured
             network layer interface.";
        }
      }
      container control-plane-protocols {
        description
          "Container for the list of routing protocol instances.";
        list control-plane-protocol {
          key "type name";
          description
            "State data of a control plane protocol instance.

             An implementation MUST provide exactly one
             system-controlled instance of the 'direct'
             pseudo-protocol. Instances of other control plane
             protocols MAY be created by configuration.";
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            description
              "Type of the control plane protocol.";
          }
          leaf name {
            type string;
            description
              "The name of the control plane 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.

             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.

               By default, control plane protocols place their routes
               in the default RIBs.";
          }
          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;
            }
          }
          action active-route {
            description
              "Return the active RIB route that is used for the
               destination address.

               Address family specific modules MUST augment input
               parameters with a leaf named 'destination-address'.";
            output {
```

```
            container route {
              description
                "The active RIB route for the specified destination.

                 If no route exists in the RIB for the destination
                 address, no output is returned.

                 Address family specific modules MUST augment this
                 container with appropriate route contents.";
              container next-hop {
                description
                  "Route's next-hop attribute.";
                uses next-hop-state-content;
              }
              uses route-metadata;
            }
          }
        }
      }
    }

    /* Configuration Data */

    container routing {
      description
        "Configuration parameters for the routing subsystem.";
      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.";
      }
      container control-plane-protocols {
        description
          "Configuration of control plane protocol instances.";
        list control-plane-protocol {
          key "type name";
          description
            "Each entry contains configuration of a control plane
             protocol instance.";
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            description
              "Type of the control plane protocol - an identity derived
```

```
            from the 'control-plane-protocol' base identity.";
        }
        leaf name {
          type string;
          description
            "An arbitrary name of the control plane protocol
             instance.";
        }
        leaf description {
          type string;
          description
            "Textual description of the control plane protocol
             instance.";
        }
        container static-routes {
          when "derived-from-or-self(../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/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.";
            }
          }
        }
      }
    }
```

<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@2016-11-03.yang"

```
module ietf-ipv4-unicast-routing {

  yang-version "1.1";

  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:   <https://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

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

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>

     Editor:   Acee Lindem
               <mailto:acee@cisco.com>";

  description
    "This YANG module augments the 'ietf-routing' module with basic
     configuration and state data for IPv4 unicast routing.

     Copyright (c) 2016 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
     (https://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 (https://tools.ietf.org/html/rfc2119).

     This version of this YANG module is part of RFC XXXX
     (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
     full legal notices.";

  revision 2016-11-03 {
    description
```

```
        "Initial revision.";
      reference
        "RFC XXXX: A YANG Data Model for Routing Management";
    }

    /* Identities */

    identity ipv4-unicast {
      base rt:ipv4;
      description
        "This identity represents the IPv4 unicast address family.";
    }

    /* State data */

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
      when "derived-from-or-self(../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "This leaf augments an IPv4 unicast route.";
      leaf destination-prefix {
        type inet:ipv4-prefix;
        description
          "IPv4 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in IPv4 unicast routes.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
```

```
          + "rt:next-hop-list/rt:next-hop" {
        when "derived-from-or-self(../../../../../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast.";
        }
        description
          "This leaf augments the 'next-hop-list' case of IPv4 unicast
           routes.";
        leaf address {
          type inet:ipv4-address;
          description
            "IPv4 address of the next-hop.";
        }
      }

      augment
        "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
        when "derived-from-or-self(../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast RIBs.";
        }
        description
          "This augment adds the input parameter of the 'active-route'
           action.";
        leaf destination-address {
          type inet:ipv4-address;
          description
            "IPv4 destination address.";
        }
      }

      augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
            + "rt:output/rt:route" {
        when "derived-from-or-self(../../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast.";
        }
        description
          "This augment adds the destination prefix to the reply of the
           'active-route' action.";
        leaf destination-prefix {
          type inet:ipv4-prefix;
          description
            "IPv4 destination prefix.";
        }
```

```
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'next-hop-list' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    /* Configuration data */

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-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/simple-next-hop" {
                 description
                   "Augment 'simple-next-hop' case in IPv4 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next-hop.";
                 }
               }
               augment "next-hop-options/next-hop-list/next-hop-list/"
                     + "next-hop" {
                 description
                   "Augment 'next-hop-list' case in IPv4 static
                    routes.";
                 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@2016-11-03.yang"

```
module ietf-ipv6-unicast-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";

  prefix "v6ur";

  import ietf-routing {
    prefix "rt";
  }

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

  include ietf-ipv6-router-advertisements {
    revision-date 2016-11-03;
  }

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

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

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

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

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>

     Editor:   Acee Lindem
               <mailto:acee@cisco.com>";
```

```
     description
       "This YANG module augments the 'ietf-routing' module with basic
        configuration and state data for IPv6 unicast routing.

        Copyright (c) 2016 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
        (https://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 (https://tools.ietf.org/html/rfc2119).

        This version of this YANG module is part of RFC XXXX
        (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
        full legal notices.";

     revision 2016-11-03 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* Identities */

     identity ipv6-unicast {
       base rt:ipv6;
       description
         "This identity represents the IPv6 unicast address family.";
     }

     /* State data */

     augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
       when "derived-from-or-self(../../rt:address-family, "
          + "'v6ur:ipv6-unicast')" {
         description
           "This augment is valid only for IPv6 unicast.";
       }
       description
         "This leaf augments an IPv6 unicast route.";
```

```
      leaf destination-prefix {
        type inet:ipv6-prefix;
        description
          "IPv6 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in IPv6 unicast routes.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
          + "rt:next-hop-list/rt:next-hop" {
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This leaf augments the 'next-hop-list' case of IPv6 unicast
         routes.";
      leaf address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment
      "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
      when "derived-from-or-self(../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast RIBs.";
      }
```

```
      description
        "This augment adds the input parameter of the 'active-route'
         action.";
      leaf destination-address {
        type inet:ipv6-address;
        description
          "IPv6 destination address.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route" {
      when "derived-from-or-self(../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This augment adds the destination prefix to the reply of the
         'active-route' action.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        description
          "IPv6 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
```

```
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'next-hop-list' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    /* Configuration data */

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-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/simple-next-hop" {
                description
                  "Augment 'simple-next-hop' case in IPv6 static
```

```
                   routes.";
                 leaf next-hop-address {
                   type inet:ipv6-address;
                   description
                     "IPv6 address of the next-hop.";
                 }
               }
               augment "next-hop-options/next-hop-list/next-hop-list/"
                     + "next-hop" {
                 description
                   "Augment 'next-hop-list' case in IPv6 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv6-address;
                   description
                     "IPv6 address of the next-hop.";
                 }
               }
             }
           }
         }
       }
     }
   }
```

<CODE ENDS>

9.1. IPv6 Router Advertisements Submodule

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-router-advertisements@2016-11-03.yang"

```
submodule ietf-ipv6-router-advertisements {

  yang-version "1.1";

  belongs-to ietf-ipv6-unicast-routing {
    prefix "v6ur";
  }

  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:   <https://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>

       WG Chair: Lou Berger
                 <mailto:lberger@labn.net>

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

       Editor:   Ladislav Lhotka
                 <mailto:lhotka@nic.cz>

       Editor:   Acee Lindem
                 <mailto:acee@cisco.com>";

    description
      "This YANG module augments the 'ietf-ip' module with
       configuration and state data of IPv6 router advertisements.

       Copyright (c) 2016 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
       (https://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 (https://tools.ietf.org/html/rfc2119).

       This version of this YANG module is part of RFC XXXX
       (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
       full legal notices.";
```

```
     reference
       "RFC 4861: Neighbor Discovery for IP version 6 (IPv6).";

     revision 2016-11-03 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* State data */

     augment "/if:interfaces-state/if:interface/ip:ipv6" {
       description
         "Augment interface state data with parameters of IPv6 router
          advertisements.";
       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.";
            }
          }
        }
      }
    }

    /* Configuration data */

    augment "/if:interfaces/if:interface/ip:ipv6" {
      description
        "Augment interface configuration with parameters of IPv6 router
         advertisements.";
      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.";
                 }
               }
             }
           }
         }
       }
     }
   }
```

<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
   ---------------------------------------------------------------------
```

This document registers the following YANG submodule in the YANG Module Names registry [RFC6020]:

```
   ---------------------------------------------------------------------
   name:         ietf-ipv6-router-advertisements
   parent:       ietf-ipv6-unicast-routing
   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 a management protocol with secure transport layer, such as NETCONF [RFC6241]. 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 configuration data nodes defined in the YANG modules belonging to 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" in NETCONF, 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:

/routing/control-plane-protocols/control-plane-protocol: This list specifies the control plane protocols configured on a device.

/routing/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, Yingzhen Qu, 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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

13.2. Informative References

[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>.

[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>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

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 router-id?                yang:dotted-quad
      +--rw control-plane-protocols
      |  +--rw control-plane-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 (v6ur:next-hop-options)
      |        |           +--:(v6ur:simple-next-hop)
      |        |           |  +--rw v6ur:outgoing-interface?
      |        |           |  +--rw v6ur:next-hop-address?
      |        |           +--:(v6ur:special-next-hop)
      |        |           |  +--rw v6ur:special-next-hop?   enumeration
      |        |           +--:(v6ur:next-hop-list)
      |        |              +--rw v6ur:next-hop-list
      |        |                 +--rw v6ur:next-hop* [index]
      |        |                    +--rw v6ur:index              string
      |        |                    +--rw v6ur:outgoing-interface?
      |        |                    +--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 (v4ur:next-hop-options)
      |                    +--:(v4ur:simple-next-hop)
      |                    |  +--rw v4ur:outgoing-interface?
      |                    |  +--rw v4ur:next-hop-address?
      |                    +--:(v4ur:special-next-hop)
      |                    |  +--rw v4ur:special-next-hop?   enumeration
      |                    +--:(v4ur:next-hop-list)
      |                       +--rw v4ur:next-hop-list
      |                          +--rw v4ur:next-hop* [index]
      |                             +--rw v4ur:index              string
      |                             +--rw v4ur:outgoing-interface?
      |                             +--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 router-id?                yang:dotted-quad
|  +--ro interfaces
|  |  +--ro interface*   if:interface-state-ref
|  +--ro control-plane-protocols
|  |  +--ro control-plane-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)
|        |     |     +--:(simple-next-hop)
|        |     |     |  +--ro outgoing-interface?
|        |     |     |  +--ro v6ur:next-hop-address?
|        |     |     |  +--ro v4ur:next-hop-address?
|        |     |     +--:(special-next-hop)
|        |     |     |  +--ro special-next-hop?        enumeration
|        |     |     +--:(next-hop-list)
|        |     |        +--ro next-hop-list
|        |     |           +--ro next-hop*
|        |     |              +--ro outgoing-interface?
|        |     |              +--ro v6ur:address?
|        |     |              +--ro v4ur: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
|        +---x active-route
|           +---w input
|           |  +---w v6ur:destination-address?   inet:ipv6-address
|           |  +---w v4ur:destination-address?   inet:ipv4-address
|           +--ro output
|              +--ro route
|                 +--ro next-hop
|                 |  +--ro (next-hop-options)
|                 |     +--:(simple-next-hop)
|                 |     |  +--ro outgoing-interface?
|                 |     |  +--ro v6ur:next-hop-address?
```

```
|                |     |  +--ro v4ur:next-hop-address?
|                |     +--:(special-next-hop)
|                |     |  +--ro special-next-hop?     enumeration
|                |     +--:(next-hop-list)
|                |        +--ro next-hop-list
|                |           +--ro next-hop*
|                |              +--ro outgoing-interface?
|                |              +--ro v6ur: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 does not support the feature "multiple-ribs". This means that a single system-controlled RIB is available for each supported address family - IPv4, IPv6 or both. These RIBs are also 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.

For hosts that are never intended to act as routers, the ability to turn on sending IPv6 router advertisements (Section 5.4) should be removed.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" control plane protocol instances.

Appendix C. Example: Adding a New Control Plane Protocol

This appendix demonstrates how the core routing data model can be extended to support a new control plane 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 5.3.2.

```
   module example-rip {

     yang-version "1.1";

     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:ribs/rt:rib/rt:routes/rt:route" {
       when "derived-from-or-self(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:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route" {
      description
        "RIP-specific route attributes in the output of 'active-route'
         RPC.";
      uses route-content;
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol" {
      when "derived-from-or-self(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. Data Tree Example

This section contains an example instance data tree in the JSON encoding [RFC7951], containing both configuration and state data. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "c2e1f54169aa7f36e1a6e8d0865d441d3600f9c4",
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2016-11-03",
        "feature": [
          "multiple-ribs",
          "router-id"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
        "conformance-type": "implement"
      },
      {
```

```
          "name": "ietf-interfaces",
          "revision": "2014-05-08",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-inet-types",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "ietf-yang-types",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "iana-if-type",
          "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
          "revision": "",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        }
      ]
    }
  }
```

A simple network set-up as shown in Figure 3 is assumed: 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.

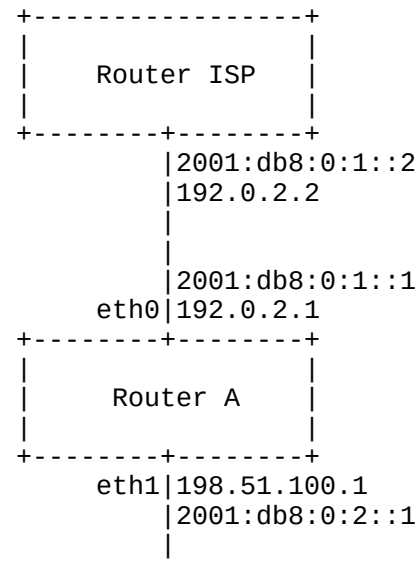


Figure 3: Example network configuration

The instance data tree could then be as follows:

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ],
          "forwarding": true
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "2001:0db8:0:1::1",
              "prefix-length": 64
            }
          ],
          "forwarding": true,
```

```
            "autoconf": {
              "create-global-addresses": false
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "description": "Interface to the internal network.",
          "ietf-ip:ipv4": {
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ],
            "forwarding": true
          },
          "ietf-ip:ipv6": {
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "forwarding": true,
            "autoconf": {
              "create-global-addresses": false
            },
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true
            }
          }
        }
      ]
    },
    "ietf-interfaces:interfaces-state": {
      "interface": [
        {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:E9",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:27+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
```

```
            "mtu": 1500,
            "address": [
              {
                "ip": "192.0.2.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true,
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:EA",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:29+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
```

```
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true,
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        }
      ]
    },
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0",
            "description":
              "Static routing is used for the internal network.",
            "static-routes": {
              "ietf-ipv4-unicast-routing:ipv4": {
                "route": [
                  {
                    "destination-prefix": "0.0.0.0/0",
                    "next-hop": {
                      "next-hop-address": "192.0.2.2"
                    }
                  }
                ]
              },
              "ietf-ipv6-unicast-routing:ipv6": {
                "route": [
                  {
                    "destination-prefix": "::/0",
                    "next-hop": {
                      "next-hop-address": "2001:db8:0:1::2"
```

```
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    },
    "ietf-routing:routing-state": {
      "interfaces": {
        "interface": [
          "eth0",
          "eth1"
        ]
      },
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0"
          }
        ]
      },
      "ribs": {
        "rib": [
          {
            "name": "ipv4-master",
            "address-family":
              "ietf-ipv4-unicast-routing:ipv4-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "192.0.2.1/24",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "route-preference": 0,
                  "source-protocol": "ietf-routing:direct",
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "198.51.100.0/24",
                  "next-hop": {
                    "outgoing-interface": "eth1"
```

```
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "0.0.0.0/0",
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "next-hop": {
                    "ietf-ipv4-unicast-routing:next-hop-address":
                      "192.0.2.2"
                  },
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          },
          {
            "name": "ipv6-master",
            "address-family":
              "ietf-ipv6-unicast-routing:ipv6-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:1::/64",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:2::/64",
                  "next-hop": {
                    "outgoing-interface": "eth1"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
```

```
                    "::/0",
                  "next-hop": {
                    "ietf-ipv6-unicast-routing:next-hop-address":
                      "2001:db8:0:1::2"
                  },
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          }
        ]
      }
    }
  }
```

Appendix E. Change Log

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

E.1. Changes Between Versions -24 and -25

- Minor edits based on IETF Last Call reviews.

E.2. Changes Between Versions -23 and -24

- Fix paths in "when" expressions due to errata 4749 of RFC 7950.

E.3. Changes Between Versions -22 and -23

- Removed "route-tag" feature.
- Removed next-hop classifiers.
- Fixed invalid when expressions in augments.
- In simple-next-hop, an address, outgoing interface or both can be specified.
- RPC "fib-route" changed into RIB action "active-route".
- The requirement that direct routes be always placed in default RIBs.

E.4. Changes Between Versions -21 and -22

- Added "next-hop-list" as a new case of the "next-hop-options" choice.
- Renamed "routing protocol" to "control plane protocol" in both the YANG modules and I-D text.

E.5. Changes Between Versions -20 and -21

- Routing instances were removed.
- IPv6 RA parameters were moved to the "ietf-ipv6-router-advertisements".

E.6. Changes Between Versions -19 and -20

- Assignment of L3 interfaces to routing instances is now part of interface configuration.
- Next-hop options in configuration were aligned with state data.
- It is recommended to enclose protocol-specific configuration in a presence container.

E.7. Changes Between Versions -18 and -19

- The leaf "route-preference" was removed from the "routing-protocol" container in both "routing" and "routing-state".
- The "vrf-routing-instance" identity was added in support of a common routing-instance type in addition to the "default-routing-instance".
- Removed "enabled" switch from "routing-protocol".

E.8. Changes Between Versions -17 and -18

- The container "ribs" was moved under "routing-instance" (in both "routing" and "routing-state").
- Typedefs "rib-ref" and "rib-state-ref" were removed.
- Removed "recipient-ribs" (both state and configuration).
- Removed "connected-ribs" from "routing-protocol" (both state and configuration).

- Configuration and state data for IPv6 RA were moved under "if:interface" and "if:interface-state".

- 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).

- Specification of a default RIB is now a simple flag under "rib" (both config and state).

- Default RIBs are marked by a flag in state data.

E.9. Changes Between Versions -16 and -17

- Added Acee as a co-author.

- Removed all traces of route filters.

- Removed numeric IDs of list entries in state data.

- Removed all next-hop cases except "simple-next-hop" and "special-next-hop".

- Removed feature "multipath-routes".

- Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing form state data of an interface entry to the routing instance(s) to which the interface is assigned.

E.10. Changes Between Versions -15 and -16

- Added ’type’ as the second key component of ’routing-protocol’, both in configuration and state data.

- The restriction of no more than one connected RIB per address family was removed.

- Removed the ’id’ key of routes in RIBs. This list has no keys anymore.

- Remove the ’id’ key from static routes and make ’destination-prefix’ the only key.

- Added ’route-preference’ as a new attribute of routes in RIB.

- Added ’active’ as a new attribute of routes in RIBs.

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

E.11. Changes Between Versions -14 and -15

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

E.12. Changes Between Versions -13 and -14

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

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

- Wrote appendix about minimum implementation.
- Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.
- Extra container for the next-hop list.
- Names rather than numeric ids are used for referring to list entries in state data.

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

E.14. Changes Between Versions -11 and -12

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

E.15. Changes Between Versions -10 and -11

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

- Added nexthop-list with entries having priorities (backup) and weights (load balancing).

- Updated bibliography references.

E.16. Changes Between Versions -09 and -10

- Added subtree for state data ("/routing-state").

- Terms "system-controlled entry" and "user-controlled entry" defined and used.

- New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.

- Added grouping "router-id".

- In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".

- Renamed "main-routing-table" to "default-routing-table".

E.17. Changes Between Versions -08 and -09

- Fixed "must" expression for "connected-routing-table".

- Simplified "must" expression for "main-routing-table".

- Moved per-interface configuration of a new routing protocol under 'routing-protocol'. This also affects the 'example-rip' module.

E.18. Changes Between Versions -07 and -08

- Changed reference from RFC6021 to RFC6021bis.

E.19. Changes Between Versions -06 and -07

- The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.

- The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.

- The default behavior for IPv6 RA prefix advertisements was clarified.

- Changed type of "rt:router-id" to "ip:dotted-quad".
- Type of "rt:router-id" changed to "yang:dotted-quad".
- Fixed missing prefixes in XPath expressions.

E.20. Changes Between Versions -05 and -06

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

E.21. Changes Between Versions -04 and -05

- Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- "must" statement for "static-routes" changed to "when".
- Added "main-routing-tables" containing references to main routing tables for each address family.
- Removed the defaults for "address-family" and "safi" and made them mandatory.
- Removed the default for route-filter/type and made this leaf mandatory.
- If there is no active route for a given destination, the "active-route" RPC returns no output.
- Added "enabled" switch under "routing-protocol".

- Added "router-type" identity and "type" leaf under "router".
- Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- The "direct" pseudo-protocol is always connected to main routing tables.
- Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".
- Added "must" constraint saying that a routing table must not be its own recipient.

E.22. Changes Between Versions -03 and -04

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

E.23. Changes Between Versions -02 and -03

- Module "iana-afn-safi" moved to I-D "iana-if-type".
- Removed forwarding table.
- RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- New RPC "route-count".
- For both RPCs, specification of negative responses was added.
- Relaxed separation of router instances.
- Assignment of interfaces to router instances needn’t be disjoint.
- Route filters are now global.
- Added "allow-all-route-filter" for symmetry.

- Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- Added "router-id" leaf.
- Specified the names for IPv4/IPv6 unicast main routing tables.
- Route parameter "last-modified" changed to "age".
- Added container "recipient-routing-tables".

E.24. Changes Between Versions -01 and -02

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

E.25. Changes Between Versions -00 and -01

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

- Illegal XPath references from "get-route" to the datastore were fixed.
- Section "Security Considerations" was written.

Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Lindem
Cisco Systems

Email: acee@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2017

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

July 1, 2016

YANG Schema Mount
draft-ietf-netmod-schema-mount-02

Abstract

This document defines a mechanism to combine YANG modules into the schema defined in other YANG modules.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 2, 2017.

Table of Contents

1. Introduction

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:

- Brackets "[" and "]" enclose list keys.

- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Background

YANG has two mechanisms for extending a data model with additional nodes; "uses" and "augment". The "uses" statement explicitly incorporates the contents of a "grouping" defined in some other module. The "augment" statement explicitly adds contents to a target node defined in some other module. In both these cases, the source and/or target model explicitly defines the relationship between the models.

In some cases these mechanisms are not sufficient. For example, suppose we have a model like ietf-interfaces [RFC7223] that is defined to be implemented in a device. Now suppose we want to model a device that supports multiple logical devices [I-D.rtgyangdt-rtgwg-device-model], where each such logical device has its own instantiation of ietf-interfaces (and other models), but at the same time, we'd like to be able to manage all these logical devices from the main device. We would like something like this:

```
  +--rw interfaces
  | +--rw interface* [name]
  |    ...
  +--rw logical-device* [name]
     +--rw name             string
     |   ...
     +--rw interfaces
       +--rw interface* [name]
          ...
```

With the "uses" approach, ietf-interfaces would have to define a grouping with all its nodes, and the new model for logical devices would have to use this grouping. This is a not a scalable solution, since every time there is a new model defined, we would have to update our model for logical devices to use a grouping from the new model. Another problem is that this approach cannot handle vendor-specific modules.

With the "augment" approach, ietf-interfaces would have to augment the logical-device list with all its nodes, and at the same time

define all its nodes on the top-level. This approach is also not scalable, since there may be other models to which we would like to add the interface list.

3. Schema Mount

The schema mount mechanism defined in this document takes a different approach to the extensibility problem described in the previous section. It decouples the definition of the relation between the source and target models from the definitions of the models themselves.

This is accomplished with a YANG extension statement that is used to specify a mount point in a data model. The purpose of a mount point is to define a place in the node hierarchy where other YANG data models may be attached, without any special notation in the other YANG data models.

For each mount point supported by a server, the server populates an operational state node hierarchy with information about which models it has mounted. This node hierarchy can be read by a client in order to learn what is implemented on a server.

Schema mount applies to the data model, and specifically does not assume anything about how the mounted data is implemented. It may be implemented using the same instrumentation as the rest of the system, or it may be implemented by querying some other system. Future specifications may define mechanisms to control or monitor the implementation of specific mount points.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms, for example mounting of sub-hierarchies of a module.

3.1. Augment and Validation in Mounted Data

All paths (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) in the data models mounted at a mount point are interpreted with the mount point as the root node, and the mounted data nodes as its children. This means that data within a mounted subtree can never refer to data outside of this subtree.

3.2. Top-level RPCs

If any mounted data model defines RPCs, these RPCs can be invoked by clients by treating them as actions defined where the mount point is specified. An example of this is given in Appendix B.1.

3.3. Top-level Notifications

If the server emits a notification defined at the top-level in any mounted data model, it is treated as if the notification was attached to the data node where the mount point is specified.

4. Data Model

This document defines the YANG 1.1 module [I-D.ietf-netmod-rfc6020bis] "ietf-yang-schema-mount", which has the following structure:

```
module: ietf-yang-schema-mount
   +--ro mount-points
      +--ro mount-point* [module name]
         +--ro module                 yang:yang-identifier
         +--ro name                   yang:yang-identifier
         +--ro (data-model)
            +--:(inline-yang-library)
            |  +--ro inline-yang-library?   empty
            +--:(modules)
               +--ro modules
                  +--ro module* [name revision]
                     +--ro name                yang:yang-identifier
                     +--ro revision            union
                     +--ro schema?             inet:uri
                     +--ro namespace           inet:uri
                     +--ro feature*            yang:yang-identifier
                     +--ro deviation* [name revision]
                     |  +--ro name        yang:yang-identifier
                     |  +--ro revision    union
                     +--ro conformance-type    enumeration
                     +--ro submodule* [name revision]
                        +--ro name        yang:yang-identifier
                        +--ro revision    union
                        +--ro schema?     inet:uri
```

5. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

<CODE BEGINS> file "ietf-yang-schema-mount@2016-04-05.yang"

```
module ietf-yang-schema-mount {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
  prefix yangmnt;
```

```
    import ietf-yang-types {
      prefix yang;
      reference "RFC 6991: Common YANG Data Types";

    }
    import ietf-yang-library {
      prefix yanglib;
      reference "RFC 7895: YANG Module Library";
    }

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


    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.
    description
      "This module defines a YANG extension statement that can be used
       to incorporate data models defined in other YANG modules in a
       module.  It also defines a operational state data so that
       clients can learn which data models a server implements for the
       mount points.

       Copyright (c) 2016 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.";

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2016-07-01 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Schema Mount";
    }

    /*
     * Extension statements
     */

    extension mount-point {
      argument name;
      description
        "The argument 'name' is a yang-identifier.  The name of
         the mount point MUST be unique within the module where it
         is defined.

         The 'mount-point' statement can be present in 'anydata'.

         If a mount point is defined in a grouping, its name is bound
         to the module where the grouping is used.  Note that this
         implies that such a grouping can be used at most once in a
         module.

         A mount point defines a place in the node hierarchy where
         other data models may be attached.  A server that implements
         a module with a mount point, populates the
         /mount-points/mount-point list with detailed information on
         which data models are mounted at each mount point.

         The 'mount-yang-library' extension may be used as a
         substatement to 'mount-point'.";
    }

    extension mount-yang-library {
      description
```

```
        "The presence of this statement as a substatement to
         'mount-point' indicates that the data model defined in the
         module 'ietf-yang-library' is mounted.  When this statement is
         present, a client can discover the mounted YANG modules by
         reading from the mounted 'ietf-yang-library' data.

         This statement is useful if the mount point is defined in a
         list and different list entries may mount a different
         set of modules.";
    }

    /*
     * Operational state data nodes
     */

    container mount-points {
      config false;
      description
        "Contains information about which mount points are implemented
         in the server, and their data models.";

      list mount-point {
        key "module name";
        description
          "Contains information about which data models are implemented
           for the mountpoint 'name' defined in 'module'.";

        leaf module {
          type yang:yang-identifier;
          description
            "The name of the module where the mount point is defined.";
        }
        leaf name {
          type yang:yang-identifier;
          description
            "The name of the mount point.";
        }
        choice data-model {
          mandatory true;
          description
            "Indicates which data models the server implements
             for this mount point.

             It is expected that this choice may be augmented with other
             data model discovery mechansisms.";

          leaf inline-yang-library {
            type empty;
```

```
            description
              "This leaf indicates that the server has mounted
               'ietf-yang-library' at the mount point, and that the
               instantiation of 'ietf-yang-library' contains the
               information about which modules are mounted.

               This is useful if the mount point is defined in a
               list and different list entries may mount a different
               set of modules.";
          }

          container modules {
            description
              "The 'module' list contains the set of modules that are
               mounted at the mount point.";

            uses yanglib:module-list;
          }
        }
      }
    }
  }
```

<CODE ENDS>

6. 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: urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount

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-yang-schema-mount
namespace:   urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
prefix:      yangmnt
reference:   RFC XXXX
```

7. Security Considerations

TBD

8. Contributors

The idea of having some way to combine schemas from different YANG modules into one has been proposed independently by several groups of people: Alexander Clemm, Jan Medved, and Eric Voit ([I-D.clemm-netmod-mount]); Ladislav Lhotka ([I-D.lhotka-netmod-ysdl]); and Lou Berger and Christian Hopps.

9. References

9.1. Normative References

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-14 (work in progress), June 2016.

[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>.

[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>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

9.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Medved, J., and E. Voit, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-04 (work in progress), March 2016.

[I-D.lhotka-netmod-ysdl]
Lhotka, L., "YANG Schema Dispatching Language", draft-lhotka-netmod-ysdl-00 (work in progress), November 2015.

[I-D.rtgyangdt-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-rtgyangdt-rtgwg-device-model-04 (work in progress), May 2016.

[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>.

[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>.

Appendix A. Example: Logical Devices

Logical devices within a device typically use the same set of data models in each instance. This can be modelled with a mount point:

```
module example-logical-devices {
  yang-version 1.1;
  namespace "urn:example:logical-devices";
  prefix exld;

  import ietf-yang-schema-mount {
    prefix yangmnt;
  }

  container logical-devices {
    list logical-device {
      key name;
      leaf name {
        type string;
      }

      anydata root {
        yangmnt:mount-point logical-device;
      }
    }
  }
}
```

A server with two logical devices that both implement "ietf-interfaces" [RFC7223], "ietf-ip" [RFC7277], and "ietf-system" [RFC7317] YANG modules might populate the "mount-points" container with:

```
<mount-points
  xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount">
  <mount-point>
    <module>example-logical-devices</module>
    <name>logical-device</name>
    <modules>
      <module>
        <name>ietf-interface</name>
        <revision>2014-05-08</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-interfaces
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-ip</name>
        <revision>2014-06-16</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-ip
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-system</name>
        <revision>2014-08-06</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-system
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-yang-types</name>
        <revision>2013-07-15</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-yang-types
        </namespace>
        <conformance-type>import</conformance-type>
      </module>
    </modules>
  </mount-point>
</mount-points>
```

and the "logical-devices" container might have:

```
   <logical-devices xmlns="urn:example:logical-devices">
     <logical-device>
       <name>vrtrA</name>
       <root>
         <interfaces
             xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
           <interface>
             <name>eth0</name>
               <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                 <enabled>true</enabled>
                 ...
               </ipv6>
             ...
           </interface>
         </interfaces>
         <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
           ...
         </system>
       </root>
     </logical-device>
     <logical-device>
       <name>vrtrB</name>
       <root>
         <interfaces
             xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
           <interface>
             <name>eth0</name>
               <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                 <enabled>true</enabled>
                 ...
               </ipv6>
             ...
           </interface>
         </interfaces>
         <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
           ...
         </system>
       </root>
     </logical-device>
   </logical-devices>
```

Appendix B. Example: Network Manager

This example shows how a Network Manager application can use schema mount to define a data model with all its managed devices. Schema mount is used to mount the data models each device supports, and these data models can be discovered by a client via the "ietf-yang-library" module that is mounted for each device.

```
   module example-network-manager {
     yang-version 1.1;
     namespace "urn:example:network-manager";
     prefix exnm;

     import ietf-inet-types {
       prefix inet;
     }
     import ietf-yang-schema-mount {
       prefix yangmnt;
     }

     container managed-devices {
       description
         "The managed devices and device communication settings.";

       list device {
         key name;
         leaf name {
           type string;
         }
         container transport {
           choice protocol {
             mandatory true;
             container netconf {
               leaf address {
                 type inet:ip-address;
                 mandatory true;
               }
               container authentication {
                 // ...
               }
             }
             container restconf {
               leaf address {
                 type inet:ip-address;
                 mandatory true;
               }
               // ...
             }
           }
         }
         anydata root {
           yangmnt:mount-point managed-device {
             yangmnt:mount-yang-library;
           }
         }
       }
```

```
    }
  }
```

The "devices" container might have:

```
  <devices xmlns="urn:example:network-manager">
    <device>
      <name>rtrA</name>
      <transport>
        <netconf>
          <address>2001:db8::2</address>
          <authentication>
            ...
          </authentication>
          ...
        </netconf>
      </transport>
      <root>
        <modules-state
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
          <module>
            <name>ietf-system</name>
            ...
          </module>
        </modules-state>
        <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
          ...
        </system>
      </root>
    </device>
    <device>
      <name>rtrB</name>
      <transport>
        <restconf>
          <address>2001:db8::3</address>
          <authentication>
            ...
          </authentication>
          ...
        </restconf>
      </transport>
      <root>
        <modules-state
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
          <module>
            <name>ietf-interfaces</name>
            ...
          </module>
```

```
        </modules-state>
        <interfaces
            xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          ...
        </interfaces>
      </root>
    </device>
  </devices>
```

B.1. Invoking an RPC

A client that wants to invoke the "restart" operation [RFC7317] on the managed device "rtrA" over NETCONF [RFC6241] can send:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <managed-devices xmlns="urn:example:network-manager">
      <device>
        <name>rtrA</name>
        <root>
          <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
            <restart/>
          </system>
        </root>
      </device>
    </managed-devices>
  </action>
</rpc>
```

Appendix C. Open Issues

- Is there a use case for specifying that certain modules are required to be mounted under a mount point?

- Do we really need the case where ietf-yang-library is not mounted? The solution would be simpler if we always use ietf-yang-library at every mount point. See Appendix D.1.

- Support non-named mount points? (ysdl case) See Appendix D.2.

Appendix D. Alternative solutions

This section discusses some alternative solution ideas.

D.1. Static Mount Points with YANG Library Only

This solution supports named mount points, and always use ietf-yang-library.

There would be just one single extension statement, and no additional operational state data:

```
  extension mount-point {
    argument name;
  }
```

Data models need to be prepared with this extension:

```
  container logical-devices {
    list logical-device {
      key name;
      ...
      yangmnt:mount-point logical-device;
    }
  }
```

The tree on the server from Appendix A would look like this:

```
     "example-logical-devices:logical-devices": {
       "logical-device": [
         {
           "name": "vrtrA",
           "ietf-yang-library:modules-state": {
             "module-set-id": "ef50fe1",
             "module": [
               {
                 "name": "ietf-interfaces",
                  ...
               },
               {
                 "name": "ietf-system",
                  ...
               }
             ]
           },
           "ietf-interfaces:interfaces": {
             ...
           },
           "ietf-system:system": {
             ...
           }
         },
         {
           "name": "vrtrB",
           "ietf-yang-library:modules-state": {
             ...
           }
         }
       ]
     }
```

D.2. Dynamic Mount Points with YANG Library Only

This solution supports only non-named mount points, and always use ietf-yang-library.

There would be no extension statement. Instead, the server would populate a list of dynamic mount points. Each such mount point MUST mount ietf-yang-library.

```
     container mount-points {
       config false;
       list mount-point {
         key path;
         leaf path {
           type schema-node-path;
         }
       }
     }
```

The tree on the server from Appendix A would look like this:

```
    "ietf-yang-schema-mount:mount-points": {
      "mount-point": [
        { "path": "/exld:logical-devices/exld:logical-device" }
      ]
    },
    "example-logical-devices:logical-devices": {
      "logical-device": [
        {
          "name": "vrtrA",
          "ietf-yang-library:modules-state": {
            "module-set-id": "ef50fe1",
            "module": [
              {
                "name": "ietf-interfaces",
                 ...
              },
              {
                "name": "ietf-system",
                 ...
              }
            ]
          },
          "ietf-interfaces:interfaces": {
            ...
          },
          "ietf-system:system": {
            ...
          }
        },
        {
          "name": "vrtrB",
          "ietf-yang-library:modules-state": {
            ...
          }
        }
      ]
    }
```

A client needs to read the "/mount-points/mount-point" list in order to learn where the server has mounted data models. Next, it needs to read the "modules-state" subtree for each instantiated mount point in order to learn which modules are mounted at that instance.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: mbj@lhotka@nic.cz

NETMOD WG C. Wildes, Ed.
Internet-Draft K. Koushik, Ed.
Intended status: Standards Track Cisco Systems Inc.
Expires: January 9, 2017 July 8, 2016

Syslog YANG Model
draft-ietf-netmod-syslog-model-09

Abstract

This document describes a data model for the configuration of syslog.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2017.

Table of Contents

1. Introduction

Operating systems, processes and applications generate messages indicating their own status or the occurrence of events. These messages are useful for managing and/or debugging the network and its services. The BSD syslog protocol is a widely adopted protocol that is used for transmission and processing of the messages.

Since each process, application and operating system was written somewhat independently, there is little uniformity to the content of syslog messages. For this reason, no assumption is made upon the formatting or contents of the messages. The protocol is simply designed to transport these event messages. No acknowledgement of the receipt is made.

Essentially, a syslog process receives messages (from the kernel, processes, applications or other syslog processes) and processes those. The processing involves logging to a local file, displaying on console, user terminal, and/or relaying to syslog processes on other machines. The processing is determined by the "facility" that originated the message and the "severity" assigned to the message by the facility.

We are using definitions of syslog protocol from [RFC5424] in this RFC.

1.1. Requirements Language

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

1.2. Terminology

The term "message originator" is derived from the term "originator" as defined in [RFC5424]: an "originator" generates syslog content to be carried in a message.

The term "message distributor" is defined as a function that filters log messages and then distributes them.

The terms "relay" and "collectors" are as defined in [RFC5424].

2. Problem Statement

This document defines a YANG [RFC6020] configuration data model that may be used to configure one or more syslog processes running on a system. YANG models can be used with network management protocols such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

The data model makes use of the YANG "feature" construct which allows implementations to support only those syslog features that lie within their capabilities.

This module can be used to configure the syslog application conceptual layer [RFC5424].

3. Design of the Syslog Model

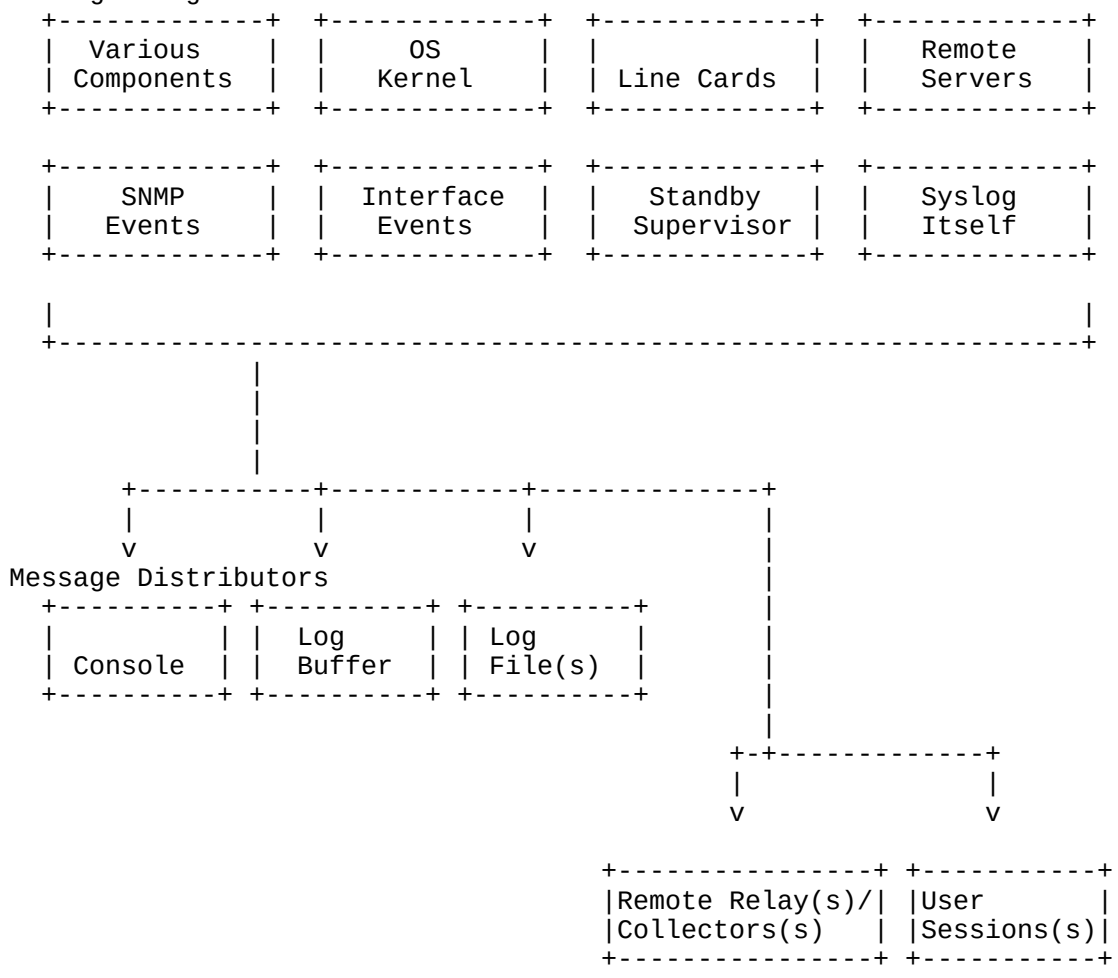
The syslog model was designed by comparing various syslog features implemented by various vendors’ in different implementations.

This draft addresses the common leafs between implementations and creates a common model, which can be augmented with proprietary features, if necessary. The base model is designed to be very simple for maximum flexibility.

Syslog consists of message originators, and message distributors. The following digram shows syslog messages flowing from a message

originator, to message distributors where suppression filtering can take place.

Many vendors extend the list of facilities available for logging in their implementation. An example is included in Extending Facilities (Appendix A.1).



The leaves in the base syslog model log-input-transports container correspond to remote message originators or remote message relays.

The leaves in the base syslog model log-actions container correspond to each message distributor:

```
   console
   log buffer
   log file(s)
   remote relay(s)/collector(s)
   user session(s).
```

Optional features are used to specified functionality that is present in specific vendor configurations.

3.1. Syslog Module

A simplified graphical representation of the complete data tree is presented here.

Each node is printed as:

<status> <flags> <name> <opts> <type> <if-features>

<status> is one of:

```
          +  for current
          x  for deprecated
          o  for obsolete
```

<flags> is one of:

```
          rw  for configuration data
          ro  for non-configuration data
          -x  for rpcs
          -n  for notifications
```

<name> is the name of the node

```
          (<name>) means that the node is a choice node
          :(<name>) means that the node is a case node
```

If the node is augmented into the tree from another module, its name is printed as <prefix>:<name>.

<opts> is one of:

```
          ?  for an optional leaf or choice
          !  for a presence container
          *  for a leaf-list or list
          [<keys>] for a list's keys
```

<type> is the name of the type for leafs and leaf-lists

If the type is a leafref, the type is printed as "-> TARGET", where TARGET is either the leafref path, with prefixed removed if possible.

<if-features> is the list of features this node depends on, printed within curly brackets and a question mark "{...}?"

```
module: ietf-syslog
   +--rw syslog
      +--rw actions
         +--rw console!
         |  +--rw log-selector
         |     +--rw (selector-facility)
         |     |  +--:(no-log-facility)
         |     |  |  +--rw no-facilities?   empty
         |     |  +--:(log-facility)
         |     |     +--rw log-facility* [facility]
         |     |        +--rw facility      union
         |     |        +--rw severity      union
         |     |        +--rw compare-op?   enumeration {select-sev-compare}?
         |     +--rw pattern-match?   string {select-match}?
         +--rw buffer
         |  +--rw log-selector
         |  |  +--rw (selector-facility)
         |  |  |  +--:(no-log-facility)
         |  |  |  |  +--rw no-facilities?   empty
         |  |  |  +--:(log-facility)
         |  |  |     +--rw log-facility* [facility]
         |  |  |        +--rw facility      union
         |  |  |        +--rw severity      union
         |  |  |        +--rw compare-op?   enumeration {select-sev-compare}?
         |  |  +--rw pattern-match?   string {select-match}?
         |  +--rw buffer-limit-bytes?      uint64 {buffer-limit-bytes}?
         |  +--rw buffer-limit-messages?   uint64 {buffer-limit-messages}?
         |  +--rw structured-data?         boolean {structured-data}?
         +--rw file
         |  +--rw log-file* [name]
         |     +--rw name               inet:uri
         |     +--rw log-selector
         |     |  +--rw (selector-facility)
         |     |  |  +--:(no-log-facility)
         |     |  |  |  +--rw no-facilities?   empty
         |     |  |  +--:(log-facility)
         |     |  |     +--rw log-facility* [facility]
         |     |  |        +--rw facility      union
         |     |  |        +--rw severity      union
         |     |  |        +--rw compare-op?   enumeration {select-sev-compare}?
         |     |  +--rw pattern-match?   string {select-match}?
         |     +--rw structured-data?   boolean {structured-data}?
```

```
|     +--rw file-archive
|        +--rw number-of-files?   uint32 {file-limit-size}?
|        +--rw max-file-size?     uint64 {file-limit-size}?
|        +--rw rollover?          uint32 {file-limit-duration}?
|        +--rw retention?         uint16 {file-limit-duration}?
+--rw remote
|  +--rw destination* [name]
|     +--rw name                    string
|     +--rw (transport)
|     |  +--:(tcp)
|     |  |  +--rw tcp
|     |  |     +--rw address?   inet:host
|     |  |     +--rw port?      inet:port-number
|     |  +--:(udp)
|     |  |  +--rw udp
|     |  |     +--rw address?   inet:host
|     |  |     +--rw port?      inet:port-number
|     |  +--:(tls)
|     |     +--rw tls
|     +--rw log-selector
|     |  +--rw (selector-facility)
|     |  |  +--:(no-log-facility)
|     |  |  |  +--rw no-facilities?   empty
|     |  |  +--:(log-facility)
|     |  |     +--rw log-facility* [facility]
|     |  |        +--rw facility      union
|     |  |        +--rw severity      union
|     |  |        +--rw compare-op?   enumeration {select-sev-compare}?
|     |  +--rw pattern-match?   string {select-match}?
|     +--rw destination-facility?   identityref
|     +--rw source-interface?       if:interface-ref
|     +--rw structured-data?        boolean {structured-data}?
|     +--rw syslog-sign! {signed-messages}?
|        +--rw cert-initial-repeat    uint16
|        +--rw cert-resend-delay      uint16
|        +--rw cert-resend-count      uint16
|        +--rw sig-max-delay          uint16
|        +--rw sig-number-resends     uint16
|        +--rw sig-resend-delay       uint16
|        +--rw sig-resend-count       uint16
+--rw session
   +--rw all-users!
   |  +--rw log-selector
   |     +--rw (selector-facility)
   |     |  +--:(no-log-facility)
   |     |  |  +--rw no-facilities?   empty
   |     |  +--:(log-facility)
   |     |     +--rw log-facility* [facility]
```

```
            |     |        +--rw facility      union
            |     |        +--rw severity      union
            |     |        +--rw compare-op?   enumeration {select-sev-compare}?
            |     +--rw pattern-match?   string {select-match}?
            +--rw user* [name]
               +--rw name            string
               +--rw log-selector
                  +--rw (selector-facility)
                  |  +--:(no-log-facility)
                  |  |  +--rw no-facilities?   empty
                  |  +--:(log-facility)
                  |     +--rw log-facility* [facility]
                  |        +--rw facility      union
                  |        +--rw severity      union
                  |        +--rw compare-op?   enumeration {select-sev-compare}?
                  +--rw pattern-match?   string {select-match}?
```

4. Syslog YANG Modules

4.1. The ietf-syslog-types Module

This module references [RFC5424].

```
<CODE BEGINS> file "ietf-syslog-types.yang"
module ietf-syslog-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog-types";
  prefix syslogtypes;

  organization "IETF NETMOD (NETCONF Data Modeling Language) Working
                Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

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

     Editor:   Kiran Agrahara Sreenivasa
               <mailto:kkoushik@cisco.com>

     Editor:   Clyde Wildes
               <mailto:cwildes@cisco.com>";
  description
    "This module contains a collection of YANG type definitions for
     SYSLOG.
```

```
  reference
    "RFC 5424: The Syslog Protocol";

  revision 2016-07-08 {
    description
      "Initial Revision";
    reference
      "RFC XXXX: SYSLOG YANG Model";
  }

  typedef severity {
    type enumeration {
      enum "emergency" {
        value 0;
        description
          "Emergency Level Msg";
      }
      enum "alert" {
        value 1;
        description
          "Alert Level Msg";
      }
      enum "critical" {
        value 2;
        description
          "Critical Level Msg";
      }
      enum "error" {
        value 3;
```

```
        description
          "Error Level Msg";
      }
      enum "warning" {
        value 4;
        description
          "Warning Level Msg";
      }
      enum "notice" {
        value 5;
        description
          "Notification Level Msg";
      }
      enum "info" {
        value 6;
        description
          "Informational Level Msg";
      }
      enum "debug" {
        value 7;
        description
          "Debugging Level Msg";
      }
    }
    description
      "The definitions for Syslog message severity as per RFC 5424.";
  }

  identity syslog-facility {
    description
      "This identity is used as a base for all syslog facilities as
       per RFC 5424.";
  }

  identity kern {
    base syslog-facility;
    description
      "The facility for kernel messages (0) as defined in RFC 5424.";
  }

  identity user {
    base syslog-facility;
    description
      "The facility for user-level messages (1) as defined in RFC 5424.";
  }

  identity mail {
    base syslog-facility;
```

```
    description
      "The facility for the mail system (2) as defined in RFC 5424.";
  }

  identity daemon {
    base syslog-facility;
    description
      "The facility for the system daemons (3) as defined in RFC 5424.";
   }

  identity auth {
    base syslog-facility;
    description
      "The facility for security/authorization messages (4) as defined
       in RFC 5424.";
  }

  identity syslog {
    base syslog-facility;
    description
      "The facility for messages generated internally by syslogd
       facility (5)as defined in RFC 5424.";
  }

  identity lpr {
    base syslog-facility;
    description
      "The facility for the line printer subsystem (6) as defined in
       RFC 5424.";
  }

  identity news {
    base syslog-facility;
    description
      "The facility for the network news subsystem (7) as defined in
       RFC 5424.";
  }

  identity uucp {
    base syslog-facility;
    description
      "The facility for the UUCP subsystem (8) as defined in RFC 5424.";
  }

  identity cron {
    base syslog-facility;
    description
      "The facility for the clock daemon (9) as defined in RFC 5424.";
```

```
  }

  identity authpriv {
    base syslog-facility;
    description
      "The facility for privileged security/authorization messages (10)
       as defined in RFC 5424.";
  }

  identity ftp {
    base syslog-facility;
    description
      "The facility for the FTP daemon (11) as defined in RFC 5424.";
  }

  identity ntp {
    base syslog-facility;
    description
      "The facility for the NTP subsystem (12) as defined in RFC 5424.";
  }

  identity audit {
    base syslog-facility;
    description
      "The facility for log audit messages (13) as defined in RFC 5424.";
  }

  identity console {
    base syslog-facility;
    description
      "The facility for log alert messages (14) as defined in RFC 5424.";
  }

  identity cron2 {
    base syslog-facility;
    description
      "The facility for the second clock daemon (15) as defined in
       RFC 5424.";
  }

  identity local0 {
    base syslog-facility;
    description
      "The facility for local use 0 messages (16) as defined in
       RFC 5424.";
  }

  identity local1 {
```

```
    base syslog-facility;
    description
      "The facility for local use 1 messages (17) as defined in
       RFC 5424.";
  }

  identity local2 {
    base syslog-facility;
    description
      "The facility for local use 2 messages (18) as defined in
       RFC 5424.";
  }

  identity local3 {
    base syslog-facility;
    description
      "The facility for local use 3 messages (19) as defined in
       RFC 5424.";
  }

  identity local4 {
    base syslog-facility;
    description
      "The facility for local use 4 messages (20) as defined in
       RFC 5424.";
  }

  identity local5 {
    base syslog-facility;
    description
      "The facility for local use 5 messages (21) as defined in
       RFC 5424.";
  }

  identity local6 {
    base syslog-facility;
    description
      "The facility for local use 6 messages (22) as defined in
       RFC 5424.";
  }

  identity local7 {
    base syslog-facility;
    description
      "The facility for local use 7 messages (23) as defined in
       RFC 5424.";
  }
}
```

<CODE ENDS>

4.2. The ietf-syslog Module

This module imports typedefs from [RFC6021] and [RFC7223], and it references [RFC5424], [RFC5425], [RFC5426], [RFC6587], and [RFC5848].

```
<CODE BEGINS> file "ietf-syslog.yang"
module ietf-syslog {
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog";
  prefix syslog;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-interfaces {
    prefix if;
  }

//import ietf-tls-client {
//  prefix tlsc;
//}

  import ietf-syslog-types {
    prefix syslogtypes;
  }

  organization "IETF NETMOD (NETCONF Data Modeling Language)
  Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

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

     Editor:   Kiran Agrahara Sreenivasa
               <mailto:kkoushik@cisco.com>

     Editor:   Clyde Wildes
               <mailto:cwildes@cisco.com>";
  description
    "This module contains a collection of YANG definitions
     for syslog configuration.
```

```
  reference
    "RFC 5424: The Syslog Protocol
     RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog
     RFC 5426: Transmission of Syslog Messages over UDP
     RFC 6587: Transmission of Syslog Messages over TCP
     RFC 5848: Signed Syslog Messages";

  revision 2016-07-08 {
    description
      "Initial Revision";
    reference
      "RFC XXXX: Syslog YANG Model";
  }

  feature buffer-limit-bytes {
    description
      "This feature indicates that local memory logging buffers
       are limited in size using a limit expressed in bytes.";
  }

  feature buffer-limit-messages {
    description
      "This feature indicates that local memory logging buffers
       are limited in size using a limit expressed in number
       of log messages.";
  }

  feature file-limit-size {
    description
```

```
      "This feature indicates that file logging resources
       are managed using size and number limits.";
  }

  feature file-limit-duration {
    description
      "This feature indicates that file logging resources
       are managed using time based limits.";
  }

  feature select-sev-compare {
    description
      "This feature represents the ability to select messages
       using the additional operators equal to, or not equal to
       when comparing the syslog message severity.";
  }

  feature select-match {
    description
      "This feature represents the ability to select messages based
       on a Posix 1003.2 regular expression pattern match.";
  }

  feature structured-data {
    description
      "This feature represents the ability to log messages
       in structured-data format as per RFC 5424.";
  }

  feature signed-messages {
    description
      "This feature represents the ability to configure signed
       syslog messages according to RFC 5848.";
  }

  grouping log-severity {
    description
      "This grouping defines the severity value that is used to
       select log messages.";
    leaf severity {
      type union {
        type syslogtypes:severity;
        type enumeration {
          enum all {
            value -1;
            description
              "This enum describes the case where all severities
               are selected.";
```

```
          }
          enum none {
            value -2;
            description
              "This enum describes the case where no severities
               are selected.";
          }
        }
      }
      mandatory true;
      description
        "This leaf specifies the syslog message severity. When
         severity is specified, the default severity comparison
         is all messages of the specified severity and greater are
         selected. 'all' is a special case which means all severities
         are selected. 'none' is a special case which means that
         no selection should occur or disable this filter.";
    }
    leaf compare-op {
      when '../severity != "all" and
           ../severity != "none"' {
        description
          "The compare-op is not applicable for severity 'all' or
           severity 'none'";
      }
      if-feature select-sev-compare;
      type enumeration {
        enum equals-or-higher {
          description
            "This enum specifies all messages of the specified
             severity and higher are logged according to the
             given log-action";
        }
        enum equals {
          description
            "This enum specifies all messages that are for
             the specified severity are logged according to the
             given log-action";
        }
        enum not-equals {
          description
            "This enum specifies all messages that are not for
             the specified severity are logged according to the
             given log-action";
        }
      }
      default equals-or-higher;
      description
```

```
        "This leaf describes the option to specify how the
         severity comparison is performed.";
    }
  }

  grouping selector {
    description
      "This grouping defines a syslog selector which is used to
       select log messages for the log-action (buffer, file,
       etc). Choose one of the following:
         no-log-facility
         log-facility [<facility> <severity>...]";
    container log-selector {
      description
        "This container describes the log selector parameters
         for syslog.";
      choice selector-facility {
        mandatory true;
        description
          "This choice describes the option to specify no
           facilities, or a specific facility which can be
           all for all facilities.";
        case no-log-facility {
          description
            "This case specifies no facilities will match when
             comparing the syslog message facility. This is a
             method that can be used to effectively disable a
             particular log-action (buffer, file, etc).";
          leaf no-facilities {
            type empty;
            description
            "This leaf specifies that no facilities are selected
             for this log-action.";
          }
        }
        case log-facility {
          description
            "This case specifies one or more specified facilities
             will match when comparing the syslog message facility.";
          list log-facility {
            key facility;
            description
              "This list describes a collection of syslog
               facilities and severities.";
            leaf facility {
              type union {
                type identityref {
                  base syslogtypes:syslog-facility;
```

```
                }
                type enumeration {
                  enum all {
                    description
                      "This enum describes the case where all
                       facilities are requested.";
                  }
                }
              }
              description
                "The leaf uniquely identifies a syslog facility.";
            }
            uses log-severity;
          }
        }
      }
      leaf pattern-match {
        if-feature select-match;
        type string;
        description
          "This leaf desribes a Posix 1003.2 regular expression
           string that can be used to select a syslog message for
           logging. The match is performed on the RFC 5424
           SYSLOG-MSG field.";
      }
    }
  }

  grouping structured-data {
    description
      "This grouping defines the syslog structured data option
       which is used to select the format used to write log
       messages.";
    leaf structured-data {
      if-feature structured-data;
      type boolean;
      default false;
      description
        "This leaf describes how log messages are written to
         the log file. If true, messages will be written
         with one or more STRUCTURED-DATA elements as per
         RFC5424; if false, messages will be written with
         STRUCTURED-DATA = NILVALUE.";
    }
  }

  container syslog {
    description
```

```
      "This container describes the configuration parameters for
       syslog.";
    container actions {
      description
        "This container describes the log-action parameters
         for syslog.";
      container console {
        presence "Enables logging console configuration";
        description
          "This container describes the configuration parameters for
           console logging.";
        uses selector;
      }
      container buffer {
        description
          "This container describes the configuration parameters for
           local memory buffer logging. The buffer is circular in
           nature, so newer messages overwrite older messages after
           the buffer is filled. The method used to read syslog messages
           from the buffer is supplied by the local implementation.";
        uses selector;
        leaf buffer-limit-bytes {
          if-feature buffer-limit-bytes;
          type uint64;
          units "bytes";
          description
            "This leaf configures the amount of memory (in bytes) that
             will be dedicated to the local memory logging buffer.
             The default value varies by implementation.";
        }
        leaf buffer-limit-messages {
          if-feature buffer-limit-messages;
          type uint64;
          units "log messages";
          description
            "This leaf configures the number of log messages that
             will be dedicated to the local memory logging buffer.
             The default value varies by implementation.";
        }
        uses structured-data;
      }
      container file {
        description
          "This container describes the configuration parameters for
           file logging. If file-archive limits are not supplied, it
           is assumed that the local implementation defined limits will
           be used.";
        list log-file {
```

```
          key "name";
          description
            "This list describes a collection of local logging
             files.";
          leaf name {
            type inet:uri {
              pattern 'file:.*';
            }
            description
              "This leaf specifies the name of the log file which
               MUST use the uri scheme file:.";
          }
          uses selector;
          uses structured-data;
          container file-archive {
            description
              "This container describes the configuration
               parameters for log file archiving.";
            leaf number-of-files {
              if-feature file-limit-size;
              type uint32;
              description
                "This leaf specifies the maximum number of log
                 files retained. Specify 1 for implementations
                 that only support one log file.";
            }
            leaf max-file-size {
              if-feature file-limit-size;
              type uint64;
              units "megabytes";
              description
                "This leaf specifies the maximum log file size.";
            }
            leaf rollover {
              if-feature file-limit-duration;
              type uint32;
              units "minutes";
              description
                "This leaf specifies the length of time that log
                 events should be written to a specific log file.
                 Log events that arrive after the rollover period
                 cause the current log file to be closed and a new
                 log file to be opened.";
            }
            leaf retention {
              if-feature file-limit-duration;
              type uint16;
              units "hours";
```

```
            description
              "This leaf specifies the length of time that
               completed/closed log event files should be stored
               in the file system before they are deleted.";
          }
        }
      }
    }
    container remote {
      description
        "This container describes the configuration parameters for
         forwarding syslog messages to remote relays or collectors.";
      list destination {
        key "name";
        description
          "This list describes a collection of remote logging
           destinations.";
        leaf name {
          type string;
          description
            "An arbitrary name for the endpoint to connect to.";
        }
        choice transport {
          mandatory true;
          description
            "This choice describes the transport option.";
          case tcp {
            container tcp {
              description
                "This container describes the TCP transport
                 options.";
              reference
                "RFC 6587: Transmission of Syslog Messages over TCP";
              leaf address {
                type inet:host;
                description
                  "The leaf uniquely specifies the address of
                   the remote host. One of the following must
                   be specified: an ipv4 address, an ipv6
                   address, or a host name.";
              }
              leaf port {
                type inet:port-number;
                default 514;
                description
                  "This leaf specifies the port number used to
                   deliver messages to the remote server.";
              }
```

```
              }
            }
            case udp {
              container udp {
                description
                  "This container describes the UDP transport
                   options.";
                reference
                  "RFC 5426: Transmission of Syslog Messages over UDP";
                leaf address {
                  type inet:host;
                  description
                    "The leaf uniquely specifies the address of
                     the remote host. One of the following must be
                     specified: an ipv4 address, an ipv6 address,
                     or a host name.";
                }
                leaf port {
                  type inet:port-number;
                  default 514;
                  description
                    "This leaf specifies the port number used to
                     deliver messages to the remote server.";
                }
              }
            }
            case tls {
              container tls {
                description
                  "This container describes the TLS transport options.";
                reference
                  "RFC 5425: Transport Layer Security (TLS) Transport
                   Mapping for Syslog ";
//              uses tlsc:initiating-tls-client-grouping {
//                refine port {
//                  default 6514;
//                  description
//                    "TCP port 6514 has been allocated as the default
//                     port for syslog over TLS.";
//                }
//              }
              }
            }
          }
          uses selector;
          leaf destination-facility {
            type identityref {
              base syslogtypes:syslog-facility;
```

```
            }
            default syslogtypes:local7;
            description
              "This leaf specifies the facility used in messages
               delivered to the remote server.";
          }
          leaf source-interface {
            type if:interface-ref;
            description
              "This leaf sets the source interface for the remote
               syslog server. Either the interface name or the
               interface IP address can be specified. If not set,
               messages sent to a remote syslog server will
               contain the IP address of the interface the syslog
               message uses to exit the network element";
          }
          uses structured-data;
          container syslog-sign {
            if-feature signed-messages;
            presence
              "If present, syslog-sign is activated.";
            description
              "This container describes the configuration
               parameters for signed syslog messages as described
               by RFC 5848.";
            reference
              "RFC 5848: Signed Syslog Messages";
            leaf cert-initial-repeat {
              type uint16;
              mandatory true;
              description
              "This leaf specifies the number of times each
               Certificate Block should be sent before the first
               message is sent.";
            }
            leaf cert-resend-delay {
              type uint16;
              mandatory true;
              description
                "This leaf specifies the maximum time delay in
                 seconds until resending the Certificate Block.";
            }
            leaf cert-resend-count {
              type uint16;
              mandatory true;
              description
                "This leaf specifies the maximum number of other
                 syslog messages to send until resending the
```

```
                Certificate Block.";
            }
            leaf sig-max-delay {
              type uint16;
              mandatory true;
              description
                "This leaf specifies when to generate a new
                 Signature Block. If this many seconds have
                 elapsed since the message with the first message
                 number of the Signature Block was sent, a new
                 Signature Block should be generated.";
            }
            leaf sig-number-resends {
              type uint16;
              mandatory true;
              description
                "This leaf specifies the number of times a
                 Signature Block is resent. (It is recommended to
                 select a value of greater than 0 in particular
                 when the UDP transport [RFC5426] is used.).";
            }
            leaf sig-resend-delay {
              type uint16;
              mandatory true;
              description
                "This leaf specifies when to send the next
                 Signature Block transmission based on time. If
                 this many seconds have elapsed since the previous
                 sending of this Signature Block, resend it.";
            }
            leaf sig-resend-count {
              type uint16;
              mandatory true;
              description
                "This leaf specifies when to send the next
                 Signature Block transmission based on a count.
                 If this many other syslog messages have been sent
                 since the previous sending of this Signature
                 Block, resend it.";
            }
          }
        }
      }
      container session {
        description
          "This container describes the configuration parameters for
           user CLI session logging configuration.";
        container all-users {
```

```
          presence "Enables logging to all user sessions.";
          description
            "This container describes the configuration
             parameters for all users.";
          uses selector;
        }
        list user {
          key "name";
          description
            "This list describes a collection of user names.";
          leaf name {
            type string;
            description
              "This leaf uniquely describes a user name which
               is the login name of the user whose session
               is to receive log messages.";
          }
          uses selector;
        }
      }
    }
  }
}
<CODE ENDS>
```

5. Usage Examples

Requirement:
Enable console logging of syslogs of severity critical

Here is the example syslog configuration xml:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog"
              xmlns:syslog="urn:ietf:params:xml:ns:yang:ietf-syslog">
        <actions>
          <console>
            <log-selector>
              <log-facility>
                <facility>all</facility>
                <severity>critical</severity>
              </log-facility>
            </log-selector>
          </console>
```

```
          </actions>
        </syslog>
      </config>
    </edit-config>
  </rpc>

  <?xml version="1.0" encoding="UTF-8"?>
  <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
  </rpc-reply>
```

Enable remote logging of syslogs to udp destination 2001:db8:a0b:12f0::1 for facility auth, severity error

```
  <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
      <target>
        <candidate/>
      </target>
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog"
                xmlns:syslog="urn:ietf:params:xml:ns:yang:ietf-syslog">
          <actions>
            <remote>
              <destination>
                <name>remote1</name>
                <udp>
                  <address>2001:db8:a0b:12f0::1</address>
                </udp>
                <log-selector>
                  <log-facility>
                    <facility xmlns:syslogtypes=
                      "urn:ietf:params:xml:ns:yang:ietf-syslog-types">
                      syslogtypes:auth</facility>
                    <severity>error</severity>
                  </log-facility>
                </log-selector>
              </destination>
            </remote>
          </actions>
        </syslog>
      </config>
    </edit-config>
  </rpc>


  <?xml version="1.0" encoding="UTF-8"?>
  <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
<ok/>
</rpc-reply>
```

6. Acknowledgements

The authors wish to thank the following who commented on this proposal:

Martin Bjorklund
Jim Gibson
Jeffrey Haas
John Heasley
Giles Heron
Lisa Huang
Mahesh Jethanandani
Jeffrey K Lange
Jan Lindblad
Chris Lonvick
Tom Petch
Juergen Schoenwaelder
Jason Sterne
Peter Van Horne
Bert Wijnen
Aleksandr Zhdankin

7. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC3688].

Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-syslog-types

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-syslog-types namespace: urn:ietf:params:xml:ns:yang:ietf-syslog-types

prefix: ietf-syslog-types reference: RFC XXXX

Following the format in RFC 3688, the following registration is requested to be made:

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

Registrant Contact: The IESG.

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

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

name: ietf-syslog namespace: urn:ietf:params:xml:ns:yang:ietf-syslog

prefix: ietf-syslog

reference: RFC XXXX

8. Security Considerations

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

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

8.1. Resource Constraints

Network administrators must take the time to estimate the appropriate memory limits caused by the configuration of actions/buffer using buffer-limit-bytes and/or buffer-limit-messages where necessary to limit the amount of memory used.

Network administrators must take the time to estimate the appropriate storage capacity caused by the configuration of actions/file using file-archive attributes to limit storage used.

It is the responsibility of the network admisintrator to ensure that the configured message flow does not overwhelm system resources.

8.2. Inappropriate Configuration

It is the responsibility of the network admisintrator to ensure that the messages are actually going to the intended recipients.

9. References

9.1. 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>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[RFC5425] Miao, F., Ed., Ma, Y., Ed., and J. Salowey, Ed., "Transport Layer Security (TLS) Transport Mapping for Syslog", RFC 5425, DOI 10.17487/RFC5425, March 2009, <http://www.rfc-editor.org/info/rfc5425>.

[RFC5426] Okmianski, A., "Transmission of Syslog Messages over UDP", RFC 5426, DOI 10.17487/RFC5426, March 2009, <http://www.rfc-editor.org/info/rfc5426>.

[RFC5848] Kelsey, J., Callas, J., and A. Clemm, "Signed Syslog Messages", RFC 5848, DOI 10.17487/RFC5848, May 2010, <http://www.rfc-editor.org/info/rfc5848>.

[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>.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <http://www.rfc-editor.org/info/rfc6021>.

[RFC6587] Gerhards, R. and C. Lonvick, "Transmission of Syslog Messages over TCP", RFC 6587, DOI 10.17487/RFC6587, April 2012, <http://www.rfc-editor.org/info/rfc6587>.

[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>.

9.2. Informative 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>.

[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>.

Appendix A. Implementor Guidelines

A.1. Extending Facilities

Many vendors extend the list of facilities available for logging in their implementation. Additional facilities may not work with the syslog protocol as defined in [RFC5424] and hence such facilities apply for local syslog-like logging functionality.

The following is an example that shows how additional facilities could be added to the list of available facilities (in this example two facilities are added):

```
   module vendor-syslog-types-example {
     namespace "urn:vendor:params:xml:ns:yang:vendor-syslog-types";
     prefix vendor-syslogtypes;

     import ietf-syslog-types {
       prefix syslogtypes;
     }

     organization "Vendor, Inc.";
     contact
       "Vendor, Inc.
        Customer Service

        E-mail: syslog-yang@vendor.com";

     description
       "This module contains a collection of vendor-sprecific YANG type
        definitions for SYSLOG.";

     revision 2016-03-20 {
       description
         "Version 1.0";
       reference
         "Vendor SYSLOG Types: SYSLOG YANG Model";
     }

     identity vendor_specific_type_1 {
       base syslogtypes:syslog-facility;
     }

     identity vendor_specific_type_2 {
       base syslogtypes:syslog-facility;
     }
   }
```

Authors' Addresses

Clyde Wildes (editor)
Cisco Systems Inc.
170 West Tasman Drive
San Jose, CA 95134
US

Phone: +1 408 527-2672
Email: cwildes@cisco.com

Kiran Koushik (editor)
Cisco Systems Inc.
12515Research Blvd., Building 4
Austin, TX 78759
US

Phone: +1 512 378-1482
Email: kkoushik@cisco.com

NETMOD
Internet-Draft
Intended status: Informational
Expires: December 15, 2017

D. Bogdanovic
Volta Networks, Inc.
B. Claise
C. Moberg
Cisco Systems, Inc.
June 13, 2017

YANG Module Classification
draft-ietf-netmod-yang-model-classification-08

Abstract

The YANG data modeling language is currently being considered for a wide variety of applications throughout the networking industry at large. Many standards development organizations (SDOs), open source software projects, vendors and users are using YANG to develop and publish YANG modules for a wide variety of applications. At the same time, there is currently no well-known terminology to categorize various types of YANG modules.

A consistent terminology would help with the categorization of YANG modules, assist in the analysis of 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 modules.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 15, 2017.

Table of Contents

1. Introduction

The Internet Engineering Steering Group (IESG) has been actively encouraging IETF working groups to use the YANG data modeling language [RFC7950], [RFC7950] and NETCONF protocol [RFC6241] 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 data modeling language in the broader industry. This extends beyond the IETF, including many standards development organizations, industry consortia, ad hoc groups, open source projects, vendors, and end-users.

There are currently no clear guidelines on how to classify the layering of YANG modules according to abstraction, or how to classify modules along the continuum spanning formal standards publications, vendor-specific modules and modules provided by end-users.

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

- The layering of modules based on their abstraction levels
- The module origin type based on the nature and intent of the content

The intent of this document is to provide a taxonomy to simplify human communication around YANG modules. While the classification boundaries are at times blurry, this document should provide a robust starting point as the YANG community gains further experience with designing and deploying modules. To be more explicit, it is expected that the classification criteria will change over time.

A number of modules have created substantial discussion during the development of this document: for examples, modules concerned with topologies. Topology modules are useful both on the Network Element level (e.g. link-state database content) as well as on the Network Service level (e.g. network-wide, configured topologies). In the end, it is the module developer that classifies the module according to the initial intent of the module content.

This document should provide benefits to multiple audiences:

- First, a common taxonomy helps with the different standards development organizations and industry consortia discussions, whose goals are determined in their respective areas of work.
- Second, operators might look at the YANG module abstraction layers to understand which Network Service YANG modules and Network Element YANG modules are available for their service composition. It is difficult to determine the module type without inspecting the YANG module itself. The YANG module name might provide some useful information but is not a definite answer. For example, an L2VPN YANG module might be a Network Service YANG module, ready to be used as a service model by a network operator. Alternatively, it might be a Network Element YANG module that contains the L2VPN data definitions required to be configured on a single device.
- And thirdly, this taxonomy would help equipment vendors (whether physical or virtual), controller vendors, orchestrator vendors to

explain to their customers the relationship between the different YANG modules they support in their products.

1.1. Terminology

[RFC7950] specifies:

- data model: A data model describes how data is represented and accessed.
- module: A YANG module defines hierarchies of schema nodes. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and "compilable".

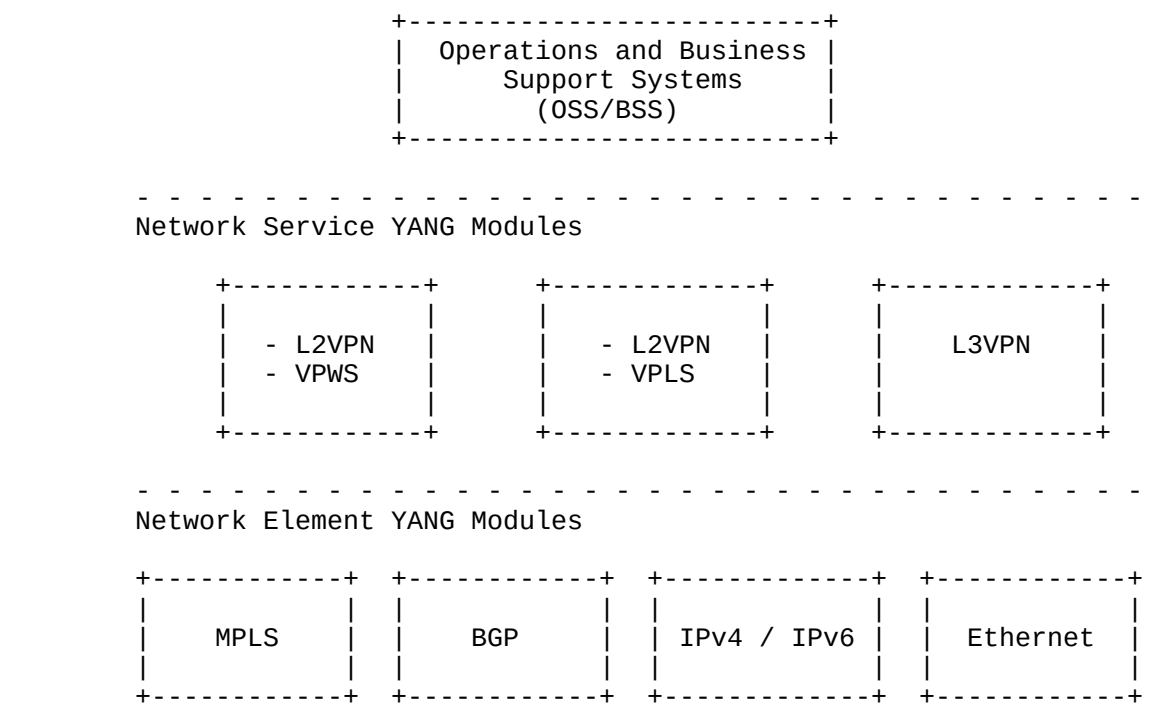
2. First Dimension: YANG Module Abstraction Layers

Module developers have taken two approaches to developing YANG modules: 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 YANG modules. This document considers both bottom-up and top-down approaches as they are both used and they each provide benefits that appeal to different groups.

For layering purposes, this document suggests the classification of YANG modules into two distinct abstraction layers:

- Network Element YANG Modules describe the configuration, state data, operations and notifications of specific device-centric technologies or features
- Network Service YANG Modules describe the configuration, state data, operations and notifications of abstract representations of services implemented on one or multiple network elements



L2VPN: Layer 2 Virtual Private Network
L3VPN: Layer 3 Virtual Private Network
VPWS: Virtual Private Wire Service
VPLS: Virtual Private LAN Service

Figure 1: YANG Module Abstraction Layers

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

For module 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 modules for e.g. routing or switching protocols requires teams that include developers with experience of implementing those protocols.

On the other hand, Network Service YANG modules are best developed by network operators experienced in defining network services for

consumption by programmers developing e.g. flow-through provisioning systems or self-service portals.

2.1. Network Service YANG Modules

Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module 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 YANG Modules 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. For the purpose of this document, the term "orchestrator" is used to describe to describe a system implementing such a process.

Network Service YANG Modules define service models to be consumed by external systems. External systems can be provisioning systems, service orchestrators, Operations Support Systems, Business Support Systems and applications exposed to network service consumers, being either internal network operations people or external customers. These modules 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 module and does not put focus on reusability of internal data definitions and groupings. The monolithic approach has the advantages of single-purpose development including development speed at the expense of reusability.

The component-based approach captures device-centric features (e.g. the definition of a VPN Routing and Forwarding (VRF), routing protocols, or packet filtering) in a vendor-independent manner. The components are designed for reuse across many service modules. 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 development 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. User-Network Interface (UNI) 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.

An example of a Network Service YANG module is in [RFC8049]. It provides an abstract model for Layer 3 IP VPN service configuration. This module includes e.g. the concept of a 'site-network-access' to represent bearer and connection parameters. An orchestrator receives operations on service instances according to the service module and decomposes the data into configuration data according to specific Network Element YANG Modules to configure the participating network elements to the service. In the case of the L3VPN module, this would include translating the 'site-network-access' parameters to the appropriate parameters in the Network Element YANG Module implemented on the constituent elements.

2.2. Network Element YANG Modules

Network Element YANG Modules describe the characteristics of a network device as defined by the vendor of that device. The modules 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 module provides a coherent data model representation of the 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 agent that implements the module.

3. Second Dimension: Module Origin Types

This document suggests classifying YANG module origin types as standard YANG modules, vendor-specific YANG modules and extensions, or user-specific YANG modules and extensions

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

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

Figure 2 illustrates the relationship between the three types of modules.

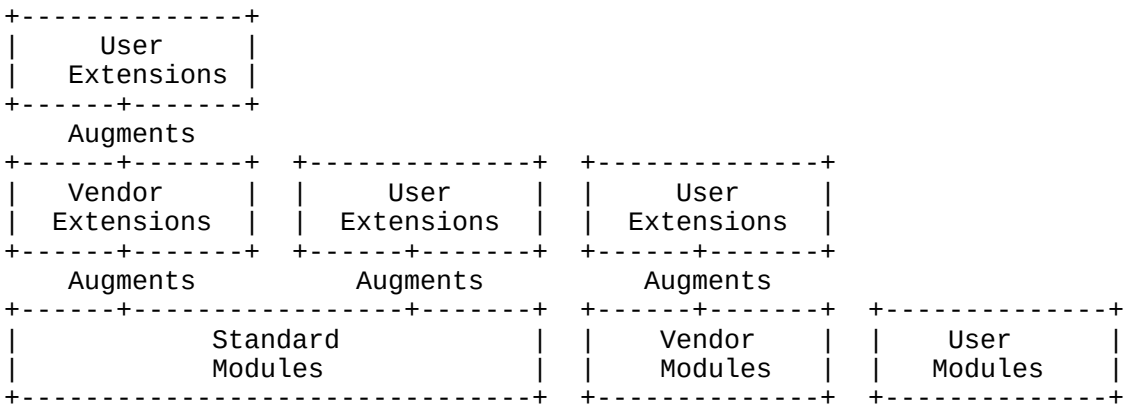


Figure 2: YANG Module Origin Types

3.1. Standard YANG Modules

Standard YANG Modules are published by standards development organizations (SDOs). Most SDOs create specifications according to a formal process in order to produce a standard that is useful for their constituencies.

The lifecycle of these modules is 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 and the IEEE.

3.2. Vendor-specific YANG Modules and Extensions

Vendor-specific YANG Modules are developed by organizations with the intent to support a specific set of implementations under control of that organization. For example vendors of virtual or physical equipment, industry consortia, and opensource projects. The intent of these modules range from providing openly published YANG modules that may eventually be contributed back to, or adopted by, an SDO, to strictly internal YANG modules not intended for external consumption.

The lifecycle of these modules 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 open source projects and SDOs in the networking industry. This includes open source projects implementing published standards as well as open source projects contributing content to SDO processes.

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

Vendors use this to extend standard modules to cover the full scope of features in implementations, which commonly is broader than that covered by the standard module.

3.3. User-specific YANG Modules and Extensions

User-specific YANG Modules are developed by organizations that operate YANG-based infrastructure including devices and orchestrators. For example, network administrators in enterprises, or at service providers. The intent of these modules is to express the specific needs for a certain implementation, above and beyond what is provided by vendors.

This module type obviously requires the infrastructure to support the introduction of user-provided modules and extensions. This would include the ability to describe the service-to-network decomposition in orchestrators and the module to configuration decomposition in devices.

The lifecycles of these modules are generally aligned with the change cadence of the infrastructure.

4. Security Considerations

This document doesn't have any Security Considerations.

5. IANA Considerations

This document has no IANA actions.

6. Acknowledgements

Thanks to David Ball and Jonathen Hansford for feedback and suggestions.

7. Change log [RFC Editor: Please remove]

version 00: Renamed and small fixes based on WG feedback.

version 01: Language fixes, collapsing of vendor data models and extensions, and the introduction of user data models and extensions.

version 02: Updated the YANG Module Catalog section, terminology alignment (YANG data model versus YANG module), explain better the distinction between the Network Element and Service YANG data models even if sometimes there are grey areas, editorial pass. Changed the use of the term 'model' to 'module' to be better aligned with RFC6020.

version 06: updates based on comments from Adrian Farrel about YANG Data Model for L3VPN Service Delivery.

version 07: updates based on comments from Pete Resnick

8. References

8.1. Normative 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>.

[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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <http://www.rfc-editor.org/info/rfc8049>.

8.2. Informative References

[I-D.ietf-netmod-acl-model] Bogdanovic, D., Koushik, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-10 (work in progress), March 2017.

[I-D.ietf-ospf-yang] Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-07 (work in progress), March 2017.

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

Authors' Addresses

Dean Bogdanovic
Volta Networks, Inc.

Email: dean@voltanet.io

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Carl Moberg
Cisco Systems, Inc.

Email: camoberg@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 13, 2017

A. Shaikh
R. Shakir
Google
K. D'Souza
AT&T

March 12, 2017

Catalog and registry for YANG models
draft-openconfig-netmod-model-catalog-02

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 September 13, 2017.

Copyright Notice

Table of Contents

1. Introduction

As YANG [RFC6020][RFC7950] 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, compatibility, and available implementations. For example, the IETF Routing Area Coordination page [RTG-AD-YANG] currently tracks over 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 sometimes 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.rtgyangdt-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 [RFC7895], and how to classify models based on their role in describing a multi-layer service [I-D.ietf-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 expressing 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 YANG modules or module bundles are available to describe a specific service or technology, and attributes of those modules 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:

- model maturity, including availability of server implementations (e.g., native device support)
- availablility of co-requisite models, and complexity of the model dependencies
- identity and reputation of the entity or organization publishing the model

The model catalog should, therefore, include key information about YANG modules, including:

- organization responsible for publishing and maintaining the module with contact information; organizations may include standards bodies (SDOs), industry forums, open source projects, individuals, etc.

- 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.; currently, identities are available for the classifications defined in [I-D.ietf-netmod-yang-model-classification]

- groupings of modules (and their versions) that are compatible with each other, and together provide a defined set of functionality

- 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

- module dependencies, e.g., a list of all of the YANG modules that are required

- pointer to the YANG module, e.g., a machine-readable URI and authentication information to allow users to verify that the model they retrieve is authentic and unaltered

- implementation information, for example, a list of available server implementations that support the module

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. Organizing YANG modules

We propose a schema for the model catalog defined using YANG (see the modules in Section 9). The YANG modules and groupings 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 release-bundles
          |     ...
          +--rw feature-bundles
          |     ...
          +--rw implementations
                ...
```

In this model, each organization publishes a list of available modules, each module having associated data describing its version, dependencies, and other basic metadata. Organizations may also publish release bundles, which are groupings of compatible modules, or feature bundles, which describes specific functionality.

3.1. Module information

Each module has several types of information associated with it. These are described below (only node names are shown).

```
+--rw modules
     +--rw module* [name version]
        +--rw name
        +--rw version
        +--rw namespace?
        +--rw prefix?
        +--rw revision?
        +--rw summary?
        +--rw classification
        |  +--rw category?
        |  +--rw subcategory?
        |  +--rw deployment-status?
        +--rw dependencies
        |  +--rw required-module*
        +--rw access
        |  +--rw uri?
        |  +--rw md5-hash?
        +--rw submodules
           +--rw submodule* [name]
              +--rw name
              +--rw access
                 +--rw uri?
                 +--rw md5-hash?
```

The basic information includes module metadata, such as its version which may be different from the YANG revision statement as in [OC-SEMVER]. Other common information includes the module's prefix, namespace, and a summary of its functionality.

The classification data includes some base information but leaves the taxonomy largely to model publishers. The category and subcategory leaves are identities that are expected to be augmented with additional values. The current version includes classification categories defined in [I-D.ietf-netmod-yang-model-classification]. 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.

Module dependencies are represented as a simple list of references to co-requisite modules indicated by 'import' statements in the module. Only the first-level dependencies are included in the list. That is, each of the listed dependences can be examined in turn to determine its dependencies.

The access data contains information required to retrieve and validate the module. Specifically, it includes a URI that can be used to download modules directly. It also includes a simple MD5 checksum to allow checking the data integrity of the module. Further

data for verifying authenticity and origin of the module may be added in future versions of the catalog.

For YANG modules that are composed of submodules, the submodules container provides their names and access information. Note that the submodules are an integral part of their parent modules, and hence are listed together with their parent and corresponding version.

4. Identifying interoperable models

YANG models for configuration and operational state data are under active development and still maturing, especially with regard to their use in production networks. As models (and their corresponding YANG modules) evolve and are revised, there is a significant challenge for users to identify the set of models that are known, or designed, to work together. This is made more complicated by the fact that models are being sourced by different organizations which may use different modeling conventions. Since there are often cross-dependencies between modules (e.g., interface configuration and various routing protocols), it is critical that users understand which modules can be used together.

The model catalog defines the notion of "release" bundles which provide a grouping of YANG modules that are part of a cohesive release. For example, a release bundle can be defined at a granular level to collect all of the modules related to interface configuration that are known to work together. These bundles can be further grouped into larger releases of models that interoperate, e.g., a release containing interoperable routing, interface, and policy-related modules.

Release bundles are also useful for implementors to know which dependencies must have what versions in order to work with a given module. For example, when an implementor wishes to support a new version of a module, the release bundle provides information about what other modules need to be upgraded in order to be compatible. It is expected that the publisher of the bundle ensures version compatibility of the release, although release bundles and the modules they include do not necessarily need to be from the same organization. We expect, however, that users and publishers of modules would be the primary source of release bundle definitions, and vendors and implementors would be the primary consumers.

4.1. Release bundle information

The schema for release bundles is shown below (only node names are shown).

```
          +--rw release-bundles
             +--rw release-bundle* [name version]
                +--rw name       string
                +--rw version    oc-cat-types:module-version-type
                +--rw members
                   +--rw member* [id]
                      +--rw id
                      +--rw type?
                      +--rw module?
                      +--rw release-bundle?
                      +--rw publisher?
                      +--rw compatible-versions*
```

The release bundle has a name and version assigned to the bundle itself, and a list of members that are part of the bundle. The list may include a reference to a module or another bundle. The compatible-versions list indicate which semantic versions [OC-SEMVER] of the respective module or bundle are known to work together.

5. Specifying functionality with feature 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 feature bundles that describe the set of schema paths required to realize a feature or service. The feature bundle paths are specified against a release bundle that ensures the paths are drawn from a set of compatible modules and/or bundles.

Feature bundles are useful for defining specific sets of functionality that can be further composed to build higher level features or services. For example, a Layer 3 VPN bundle could be composed of more specific features such as interfaces, routing, policy, and QoS. Note these bundle definitions complement the 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.

Feature bundle definitions can also be used by organizations to identify a canonical set of modules that should be used to build a

particular service. Users within the organization can be assured that the corresponding bundles are known and approved to work together to support the desired service.

Finally, a key use case for feature bundles is to define specific units of compliance for an implementation. Users can define a feature bundle containing only those paths that are required for a given usage, allowing an implementor or vendor to focus their implementation and testing on those paths rather than having to implement the entire contents of a module or bundle. The implementor may also wish to publish a deviation module that indicates which paths are not supported.

5.1. Feature bundle information

The schema for feature bundles in the catalog is shown below (note only node names are shown).

```
          +--rw feature-bundles
             +--rw feature-bundle* [name version]
                +--rw name
                +--rw version
                +--rw path*
                +--rw release-bundle
                |  +--rw name?
                |  +--rw publisher?
                |  +--rw version?
                +--rw feature-bundles
                   +--rw feature-bundle* [name]
                      +--rw name
                      +--rw publisher?
                      +--rw version?
```

Each feature bundle includes basic information such as the name of the feature or service, the bundle version, and the set of specific schema paths. The schema paths are based on the release bundle specified as part of the feature bundle. For simplicity, only one release bundle may be specified. If the schema paths in a feature bundle cross release bundle boundaries, a new release bundle should be created to include all of the paths needed by the feature bundle. The feature bundle may itself be composed of more granular feature bundles. This allows the definition of "base" features that can be reused across feature bundles.

6. Module implementations

Model implementors can use the catalog to indicate the data models they support using the implementation container. An implementation is expected to indicate a set of feature bundles it supports. The feature bundles may be defined by a user (i.e., a set of compliance units), or by the implementor or vendor to indicate the full list of what is supported.

6.1. Implementation information

The implementation information in the catalog is shown below (only node names are shown):

```
            +--rw implementations
             +--rw implementation* [id]
                +--rw id
                +--rw description?
                +--rw reference?
                +--rw platform?
                +--rw platform-version?
                +--rw status?
                +--rw feature-bundles
                   +--rw feature-bundle* [name version]
                      +--rw name
                      +--rw publisher?
                      +--rw version
```

The implementation container provides information about the platform and version on which the feature bundles are supported, as well as the status of the implementation. It also includes a URI reference to retrieve artifacts or further information on the implementation. The list of supported feature bundles are references to defined feature bundles in the catalog.

7. 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.

8. 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.

9. YANG modules

The main model catalog and associated types modules are listed below.

<CODE BEGINS> file "openconfig-catalog-types.yang"

```
module openconfig-catalog-types {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/catalog-types";

  prefix "oc-cat-types";

  import openconfig-extensions { prefix oc-ext; }


  // 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.";

  oc-ext:openconfig-version "0.2.0";

  revision "2017-03-08" {
    description
      "OpenConfig public release";
    reference "0.2.0";
  }

  revision "2016-02-15" {
    description
      "Initial OpenConfig public release";
    reference "0.1.0";
```

```
}

revision "2015-10-18" {
  description
    "Initial revision";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

identity CATALOG_MEMBER_TYPE {
  description
    "Base identity for elements in the catalog";
}

identity MODULE {
  base CATALOG_MEMBER_TYPE;
  description
    "Module elements in the catalog";
}

identity RELEASE_BUNDLE {
  base CATALOG_MEMBER_TYPE;
  description
    "Release bundle elements in the catalog";
}

identity FEATURE_BUNDLE {
  base CATALOG_MEMBER_TYPE;
  description
    "Feature bundle elements in the catalog";
}


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";
  }

  identity IETF_MODEL_LAYER {
    base MODULE_CATEGORY_BASE;
    description
      "Describes layering of models based on their abstraction
      level as defined by IETF model classification proposals";
    reference
      "IETF draft-ietf-netmod-yang-model-classification";
  }

  identity IETF_MODEL_TYPE {
    base MODULE_SUBCATEGORY_BASE;
    description
      "IETF proposed classification dimension of YANG model types as
```

```
        standard YANG models, vendor-specific, or user-specific YANG
        models and extensions";
      reference
        "IETF draft-ietf-netmod-yang-model-classification";
    }

    identity IETF_NETWORK_SERVICE {
      base IETF_MODEL_LAYER;
      description
        "Service-layer model as defined by IETF classification
        proposal";
    }

    identity IETF_NETWORK_ELEMENT {
      base IETF_MODEL_LAYER;
      description
        "Network element-layer model as defined by IETF classification
        proposal";
    }

    identity IETF_TYPE_STANDARD {
      base IETF_MODEL_TYPE;
      description
        "Models published by standards-defining organizations (SDOs)";
    }

    identity IETF_TYPE_VENDOR {
      base IETF_MODEL_TYPE;
      description
        "Developed by organizations (e.g., vendors) with the intent
        to support a specific set of implementations under control of
        that organization";
    }

    identity IETF_TYPE_USER {
      base IETF_MODEL_TYPE;
      description
        "Developed by organizations that operate YANG-based
        infrastructure including devices and orchestrators.
        The intent of these models is to express the specific needs
        for a certain implementation, above and beyond what is provided
        by vendors";
    }

    typedef module-version-type {
      type string;
      description
        "This type defines acceptable formats for the version of a
```

```
        module.  The version may be a semantic version, or a YANG
        revision statement date, and may include wildcards when
        included in a bundle compatibility list, e.g.:

        semver format: <major>.<minor>.<patch>
        examples: 0.1.0, 2.1.0, 1.1.*, 2.*.*

        revision format:  YYYY-MM-DD
        example:  2016-11-31";
    }



  }

  <CODE ENDS>


  <CODE BEGINS> file "openconfig-module-catalog.yang"

  module openconfig-module-catalog {

    yang-version "1";

    // namespace
    namespace "http://openconfig.net/yang/module-catalog";

    prefix "oc-cat";

    // import some basic types
    import openconfig-inet-types { prefix oc-inet; }
    import openconfig-catalog-types { prefix oc-cat-types; }
    import openconfig-extensions { prefix oc-ext; }


    // meta
    organization "OpenConfig working group";

    contact
      "OpenConfig working group
      www.openconfig.net";

    description
      "This module provides a schema for cataloging and descrbing
      YANG models published across various organizations.  The catalog
      contains several categories of data:

      * organizations -- entities that publish and/or maintain
```

```
        individual YANG modules or groups of modules

      * modules -- information regarding individual YANG modules,
        including their versions, dependencies, submodules, and how
        to access them

      * release bundles -- groups of modules that are compatible and
        consistent with each other (as determined by the publisher of
        of the bundle).  The release bundle does not necessarily
        correspond to a functional area, e.g., it could the entire
        set of modules published by an organization

      * feature bundles -- sets of schema paths across a
        release bundle that provide a specific set of functionality

      * implementations -- information about available module and/or
        bundle implementations and their status";

    oc-ext:openconfig-version "0.2.0";

    revision "2017-03-08" {
      description
        "OpenConfig public release";
      reference "0.2.0";
    }

    revision "2016-02-15" {
      description
        "Initial OpenConfig public release";
      reference "0.1.0";
    }

    revision "2015-10-18" {
      description
        "Initial revision";
      reference "TBD";
    }


    // grouping statements

    grouping catalog-module-common-config {
      description
        "Data definitions common for both bundles and standalone
        modules";

      leaf name {
        type string;
```

```
        description
          "The name of the module or bundle.  For modules, this
          should reflect the 'module' or 'submodule'
          statement in the YANG module file.

          For bundles, this is the canonical name for the overall
          bundle of modules which is to be released together.
          This name should be consistent over multiple
          releases";
      }

      leaf version {
        type oc-cat-types:module-version-type;
        description
          "For individual modules, this is the version number, e.g.,
          a semantic version.  The version may be the same as the date
          indicated in the module revision statement.

          For bundles, this is a semantic version number for the
          overall bundle. This version is to be defined as per the
          approach specified in the OpenConfig semantic version
          guidance - and is of the form x.y.z, where x is the major
          version, y is the minor version, and z is the patch level";
        reference
          "Semantic versioning for OpenConfig models";
      }
    }

    grouping feature-bundle-included-reference {
      description
        "References to the included feature bundles";

      leaf name {
        type leafref {
          path "../../../../../../../organizations/" +
            "organization[name=current()/../publisher]/" +
              "feature-bundles/feature-bundle/name";
        }
        description
          "Name of the referenced feature bundle";
      }

      leaf publisher {
        type leafref {
          path "../../../../../../../organizations/organization/" +
            "name";
        }
        description
```

```
            "Publisher of the referenced feature bundle";
        }

        leaf version {
          type oc-cat-types:module-version-type;
          description
            "Version of the referenced feature bundle";
        }
      }

      grouping catalog-implementation-bundle-config {
        description
          "References to the feature bundles supported by an
          implementation";

        uses feature-bundle-included-reference;
      }

      grouping catalog-implementation-bundle-top {
        description
          "Top-level grouping for the list of feature bundles
          supported by an implementation";

        container feature-bundles {
          description
            "Enclosing container for the list of feature bundles";

          list feature-bundle {
            key "name version";
            description
              "List of feature bundles supported by the implementation";

            uses catalog-implementation-bundle-config;
          }
        }
      }

      grouping catalog-implementation-config {
        description
          "Data describing any available implementations";

        leaf id {
          type string;
          description
            "An identifier for the implementation, provided by the
            implementor.  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 oc-inet:uri;
          type string;
        }
        description
          "A URI (preferred) or text reference to more detailed
          information about the implementation.";
      }


      leaf platform {
        type string;
        description
          "Name of the 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 of 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 status {
        type identityref {
          base oc-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 catalog-implementation-top {
    description
      "Top level grouping for information on model implementations";

    container implementations {
      description
        "Container for module implementation information";

      list implementation {
        key "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.";

        uses catalog-implementation-config;
        uses catalog-implementation-bundle-top;
      }
    }
  }

  grouping catalog-module-dependency-config {
    description
      "Information about module dependencies";


    leaf-list required-module {
      type leafref {
        path "../../name";
      }
      description
        "List of references to modules that are imported by the
        current module.  This list should reflect all of the 'import'
        statements in the module.";
    }
  }

  grouping catalog-module-dependency-top {
    description
      "Top-level grouping for module dependency data";

    container dependencies {
      description
        "Data about dependencies of the module";

      uses catalog-module-dependency-config;
    }
```

```
    }

    grouping catalog-module-classification-config {
      description
        "Data describing the module's classification(s)";

      leaf category {
        type identityref {
          base oc-cat-types:MODULE_CATEGORY_BASE;
        }
        description
           "Categorization of the module based on identities defined
           or used by the publishing organizations.";
      }

      leaf subcategory {
        type identityref {
          base oc-cat-types:MODULE_SUBCATEGORY_BASE;
        }
        description
           "Sub-categorization of the module based on identities
            defined or used by the publishing organizations.";
      }

      leaf deployment-status {
        type identityref {
          base oc-cat-types:MODULE_STATUS_TYPE;
        }
        description
          "Deployment status of the module -- experimental,
          standards-track, production, etc.";
      }
    }

    grouping catalog-module-classification-top {
      description
        "Data definitions related to module classfications";

      container classification {
        description
          "Container for data describing the module's classification";

        uses catalog-module-classification-config;
      }
    }

    grouping catalog-module-access-config {
      description
```

```
        "Data pertaining to retrieval and usage of the module";

      leaf uri {
        type oc-inet:uri;
        description
          "URI where module can be downloaded.  Modules may be
          made available from the catalog maintainer, or directly
          from the publisher";
      }

      leaf md5-hash {
        type string;
        description
          "Optional MD5 hash of the module file.  If specified, the
          hash may be used by users to validate data integrity";
      }
    }

    grouping catalog-module-access-top {
      description
        "Top level groupig for data related to accessing a module
        or submodule";

      container access {
        description
          "Container for data pertaining to retrieval and usage of the
          module";

        uses catalog-module-access-config;
      }
    }

    grouping catalog-module-submodule-config {
      description
        "Data definitions for submodules belonging to a
        module";

      leaf name {
        type string;
        description
          "Name of the submodule as indicated by its top-level
          'submodule' statement";
      }

    }

    grouping catalog-module-submodule-top {
      description
```

```
        "Top-level grouping for submodule information";

      container submodules {
        description
          "Data for the submodules belonging to a submodule. If the
          module does not have any submodules, this container
          should be empty.";

        list submodule {
          key "name";
          description
            "List of submodules included by a module.  All submodules
            specified by 'include' statements in the module should be
            included in this list.";

          uses catalog-module-submodule-config;
          uses catalog-module-access-top;
        }
      }
    }

    grouping catalog-module-base-config {
      description
        "Basic information describing the module, e.g., the
        YANG metadata in the module preface.";


      leaf namespace {
        type string;
        description
          "Published namespace of module, i.e., defined by the
          'namespace' ";
      }

      leaf prefix {
        type string;
        description
          "Published prefix of the module";
      }

      leaf revision {
        type string;
        description
          "Date in the revision statement of the module";
      }

      leaf summary {
        type string;
```

```
        description
          "Summary description of the module";
      }
    }

    grouping release-bundle-member-config {
      description
        "Data for each member of a bundle";

      leaf id {
        type string;
        description
          "Identifier for the bundle member";
      }

      leaf type {
        type identityref {
          base oc-cat-types:CATALOG_MEMBER_TYPE;
        }
        description
          "The type of member that is to be included within the
          release bundle. Release bundles may include modules and
          other release bundles.  Both member modules and member
          bundles should specify the list of compatible versions.";
      }

      leaf module {
        when "../type = 'oc-cat-types:MODULE'" {
          description
            "The module name is specified for bundle membrs that are
            modules";
        }
        type leafref {
          path "../../../../../../../organizations/" +
            "organization[name=current()/../publisher]/modules/" +
            "module/name";
        }
        description
          "Name of the module set which is included in this bundle -
          for example, 'openconfig-bgp'";
      }

      leaf release-bundle {
        when "../type = 'oc-cat-types:RELEASE_BUNDLE'" {
          description
            "The release bundle is specified for bundle members that
            are release bundles";
        }
```

```
        type leafref {
          path "../../../../../../../organizations/" +
            "organization[name=current()/../publisher]/" +
              "release-bundles/release-bundle/name";
        }
        description
          "Name of the module set which is included in this bundle -
          for example, 'openconfig-bgp'";
      }

      leaf publisher {
        type leafref {
          path "../../../../../../../organizations/organization/" +
            "name";
        }
        description
          "Reference to the name of the publishing organization";
      }

      leaf-list compatible-versions {
        type oc-cat-types:module-version-type;
        description
          "A list of semantic version specification of the versions
          of the specified module or release bundle which are
          compatible when building this version of the bundle.

          Version specifications may be added when changes are made
          to a module within a bundle, and this does not affect the
          interaction between it and other modules. It is expected
          that backwards compatible changes to an individual module or
          member bundle do not affect the compatibility of that
          with other members, and hence wildcard matches are allowed
          within this list.";
      }
    }

    grouping release-bundle-member-top {

      description
        "Parameters relating to models within release bundles";

      container members {
        description
          "List of bundle members which make up this release bundle. A
          member is defined as an individual YANG module specified
          in the YANG catalogue, or another release
          bundle which can be used to group multiple YANG
          models together.";
```

```
        list member {
          key "id";
          description
            "A set of modules or bundles which are part of the bundle
            of models. For example, if 'ietf-yang-types' were to be
            specified within the bundle, then this would refer to the
            individual entry within the module catalogue. If the type
            of the entry is set to bundle, then for example,
            openconfig-bgp could be referenced - which itself consists
            of separate modules.";

          uses release-bundle-member-config;

        }
      }
    }

    grouping release-bundle-top {
      description
        "Top-level container for a release bundle";

      container release-bundles {
        description
          "List of release bundles";

        list release-bundle {
          key "name version";

          description
            "List of release bundles - sets of modules and/or
            bundles which are interoperable";

          uses catalog-module-common-config;
          uses release-bundle-member-top;
        }
      }
    }

    grouping feature-bundle-release-config {
      description
        "Data definitions to identify the release bundle that the
        feature bundle is based on.";

      leaf name {
        type leafref {
          path "../../../../release-bundles/release-bundle/name";
        }
        description
```

```
          "Reference to the name of the release bundle used for the
          feature paths.";
      }

      leaf version {
        type leafref {
          path "../../../../release-bundles/" +
            "release-bundle[name=current()/../name]/version";
        }
        description
          "Reference to the release bundle version used for the
          feature paths";
      }

      leaf publisher {
        type leafref {
          path "../../../../release-bundles/" +
            "release-bundle[name=current()/../name]/publisher";
        }
        description
          "Reference to the publisher of the release bundle used for
          the feature paths";
      }
    }

    grouping feature-bundle-release-top {
      description
        "Top-level grouping for data about the release bundle used
        to specify the feature bundle";

      container release-bundle {
        description
          "Data to identify the release bundle from which the feature
          paths should be specified.  If the feature crosses
          release bundles, a new release bundle should be
          created to support the feature bundle.";

        leaf name {
          type leafref {
            path "../../../../../../organizations/" +
              "organization[name=current()/../publisher]/" +
                "release-bundles/release-bundle/name";
          }
          description
            "Name of the module set which is included in this bundle -
            for example, 'openconfig-bgp'";
        }
```

```
        leaf publisher {
          type leafref {
            path "../../../../../../organizations/organization/" +
              "name";
          }
          description
            "Reference to the name of the publishing organization";
        }

        leaf version {
          type oc-cat-types:module-version-type;
          description
            "Version of the referenced release bundle";
        }
      }
    }


    grouping feature-bundle-config {
      description
        "Data definitions for the feature bundle";

      uses catalog-module-common-config;

      leaf-list path {
        type string;
        description
          "The list of schema paths included in the feature.  The
          paths specify subtrees, i.e., all data underneath the
          specified path are included in the feature.";
      }
    }

    grouping feature-bundle-feature-config {
      description
        "Data definitions for included feature bundles";

      uses feature-bundle-included-reference;
    }

    grouping feature-bundle-feature-top {
      description
        "Top level grouping for the list of included feature
        bundles";

      container feature-bundles {
        description
          "Enclosing container for the list of included feature
```

```
          bundles.  Feature bundles may be composed from other
          smaller feature units";

        list feature-bundle {
          key "name";
          description
            "The list of feature bundles included in the current
            feature bundle.";

          uses feature-bundle-feature-config;
        }
      }

    }


    grouping feature-bundle-top {
      description
        "Top-level grouping for OpenConfig feature bundles";

      container feature-bundles {
        description
          "Enclosing container for the list of feature bundles";

        list feature-bundle {
          key "name version";
          description
            "List of feature bundles";

          uses feature-bundle-config;
          uses feature-bundle-release-top;
          uses feature-bundle-feature-top;
        }
      }
    }

    grouping catalog-module-top {
      description
        "Top level structure of the module catalog";

      container modules {
        description
          "Modules published by this organization";

        list module {
          key "name version";
          description
            "List of published modules from the organization";
```

```
          uses catalog-module-common-config;
          uses catalog-module-base-config;
          uses catalog-module-classification-top;
          uses catalog-module-dependency-top;
          uses catalog-module-access-top;
          uses catalog-module-submodule-top;
        }
      }
    }

    grouping catalog-organization-config {
      description
        "Top level grouping for data related to an organization that
        publishes module, bundles, etc.";

      leaf name {
        type string;
        description
          "Name of the maintaining organization -- the name should be
          supplied in the official format used by the organization.
          Standards Body examples:
            IETF, IEEE, MEF, ONF, etc.
          Commercial entity examples:
            AT&T, Facebook, <Vendor>
          Name of industry forum examples:
            OpenConfig, OpenDaylight, ON.Lab";
      }

      leaf type {
        type identityref {
          base oc-cat-types:ORGANIZATION_TYPE;
        }
        description
          "Type of the publishing organization";
      }

      leaf contact {
        type string;
        description
          "Contact information for the publishing organization (web
          site, email address, etc.)";
      }
    }

    grouping catalog-organization-top {
      description
        "Top level grouping for list of maintaining organizations";
```

```
      container organizations {
        description
          "List of organizations owning modules";

        list organization {
          key "name";

          description
            "List of organizations publishing YANG modules or
            module bundles";

          uses catalog-organization-config;
          uses catalog-module-top;
          uses release-bundle-top;
          uses feature-bundle-top;
          uses catalog-implementation-top;
        }
      }
    }


    grouping catalog-top {
      description
        "Top-level grouping for the YANG model catalog";

      uses catalog-organization-top;
    }

    // data definition statements

    uses catalog-top;

  }
```

<CODE ENDS>

Required extensions and types modules included below.

<CODE BEGINS> file "openconfig-extensions.yang"

```
  module openconfig-extensions {

    yang-version "1";

    // namespace
    namespace "http://openconfig.net/yang/openconfig-ext";
```

```
     prefix "oc-ext";

     // meta
     organization "OpenConfig working group";

     contact
       "OpenConfig working group
       www.openconfig.net";

     description
       "This module provides extensions to the YANG language to allow
       OpenConfig specific functionality and meta-data to be defined.";

     revision "2017-01-29" {
       description
         "Added extension for annotating encrypted values.";
       reference "TBD";
     }

     revision "2015-10-09" {
       description
         "Initial OpenConfig public release";
       reference "TBD";
     }

     revision "2015-10-05" {
       description
         "Initial revision";
       reference "TBD";
     }

     // extension statements
     extension openconfig-version {
       argument "semver" {
         yin-element false;
       }
       description
         "The OpenConfig version number for the module. This is
         expressed as a semantic version number of the form:
           x.y.z
         where:
           * x corresponds to the major version,
           * y corresponds to a minor version,
           * z corresponds to a patch version.
         This version corresponds to the model file within which it is
         defined, and does not cover the whole set of OpenConfig models.
         Where several modules are used to build up a single block of
         functionality, the same module version is specified across each
```

```
        file that makes up the module.

        A major version number of 0 indicates that this model is still
        in development (whether within OpenConfig or with industry
        partners), and is potentially subject to change.

        Following a release of major version 1, all modules will
        increment major revision number where backwards incompatible
        changes to the model are made.

        The minor version is changed when features are added to the
        model that do not impact current clients use of the model.

        The patch-level version is incremented when non-feature changes
        (such as bugfixes or clarifications to human-readable
        descriptions that do not impact model functionality) are made
        that maintain backwards compatibility.

        The version number is stored in the module meta-data.";
    }

    extension openconfig-encrypted-value {
      description
        "This extension provides an annotation on schema nodes to
        indicate that the corresponding value should be stored and
        reported in encrypted form.

        Clients reading the configuration or applied configuration
        for the node should expect to receive only the encrypted value.
        This annotation may be used on nodes such as secure passwords
        in which the device never reports a cleartext value, even
        if the input is provided as cleartext.";
    }
  }
  <CODE ENDS>


  <CODE BEGINS> file "openconfig-inet-types.yang"

  module openconfig-inet-types {

    yang-version "1";
    namespace "http://openconfig.net/yang/types/inet";
    prefix "oc-inet";

    import openconfig-extensions { prefix "oc-ext"; }

    organization
```

```
      "OpenConfig working group";

    contact
      "OpenConfig working group
      www.openconfig.net";

    description
      "This module contains a set of Internet address related
      types for use in OpenConfig modules.";

    oc-ext:openconfig-version "0.1.0";

    revision 2017-01-26 {
      description
        "Initial module for inet types";
      reference "0.1.0";
    }

    // IPv4 and IPv6 types.

    typedef ipv4-address {
      type string {
        pattern '^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|'        +
                '25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4]'  +
                '[0-9]|25[0-5])$';
      }
      description
        "An IPv4 address in dotted quad notation.";
    }

    typedef ipv6-address {
      type string {
          pattern
            // Must support compression through different lengths
            // therefore this regexp is complex.
            '^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|'         +
            '([0-9a-fA-F]{1,4}:){1,7}:|'                        +
            '([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}'         +
            '([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|' +
            '([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|' +
            '([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|' +
            '([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|' +
            '[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|'      +
            ':((:[0-9a-fA-F]{1,4}){1,7}|:)'                     +
            ')$';
      }
      description
        "An IPv6 address represented as either a full address; shortened
```

```
        or mixed-shortened formats.";
    }

    typedef ipv4-prefix {
      type string {
        pattern '^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|'       +
                '25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4]' +
                '[0-9]|25[0-5])/(([0-9])|([1-2][0-9])|(3[0-2]))$';
      }
      description
        "An IPv4 prefix represented in dotted quad notation followed by
        a slash and a CIDR mask (0 <= mask <= 32).";
    }

    typedef ipv6-prefix {
      type string {
          pattern
            '^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|'         +
            '([0-9a-fA-F]{1,4}:){1,7}:|'                        +
            '([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}'         +
            '([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|' +
            '([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|' +
            '([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|' +
            '([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|' +
            '[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|'      +
            ':((:[0-9a-fA-F]{1,4}){1,7}|:)'                     +
            ')/(12[0-8]|1[0-1][0-9]|[1-9][0-9]|[0-9])$';
      }
      description
        "An IPv6 prefix represented in full, shortened, or mixed
        shortened format followed by a slash and CIDR mask (0 <= mask <=
        128).";
    }

    typedef ip-address {
      type union {
        type ipv4-address;
        type ipv6-address;
      }
      description
        "An IPv4 or IPv6 address with no prefix specified.";
    }

    typedef ip-prefix {
      type union {
        type ipv4-prefix;
        type ipv6-prefix;
      }
```

```
      description
        "An IPv4 or IPv6 prefix.";
    }

    typedef as-number {
      type uint32;
      description
        "A numeric identifier for an autonomous system (AS). An AS is a
        single domain, under common administrative control, which forms
        a unit of routing policy. Autonomous systems can be assigned a
        2-byte identifier, or a 4-byte identifier which may have public
        or private scope. Private ASNs are assigned from dedicated
        ranges. Public ASNs are assigned from ranges allocated by IANA
        to the regional internet registries (RIRs).";
      reference
        "RFC 1930 Guidelines for creation, selection, and registration
                  of an Autonomous System (AS)
         RFC 4271 A Border Gateway Protocol 4 (BGP-4)";
    }

    typedef dscp {
      type uint8 {
        range "0..63";
      }
      description
        "A differentiated services code point (DSCP) marking within the
        IP header.";
      reference
        "RFC 2474 Definition of the Differentiated Services Field
                   (DS Field) in the IPv4 and IPv6 Headers";
    }

    typedef ipv6-flow-label {
      type uint32 {
        range "0..1048575";
      }
      description
        "The IPv6 flow-label is a 20-bit value within the IPv6 header
        which is optionally used by the source of the IPv6 packet to
        label sets of packets for which special handling may be
        required.";
      reference
        "RFC 2460 Internet Protocol, Version 6 (IPv6) Specification";
    }

    typedef port-number {
      type uint16;
      description
```

```
        "A 16-bit port number used by a transport protocol such as TCP
        or UDP.";
      reference
        "RFC 768 User Datagram Protocol
         RFC 793 Transmission Control Protocol";
    }

    typedef uri {
      type string;
      description
        "An ASCII-encoded Uniform Resource Identifier (URI) as defined
        in RFC 3986.";
      reference
        "RFC 3986 Uniform Resource Identifier (URI): Generic Syntax";
    }
  }

  <CODE ENDS>
```

10. References

10.1. 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>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

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

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

10.2. 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>.

[OC-SEMVER]
OpenConfig operator working group, "Semantic Versioning for OpenConfig models", September 2015, <http://www.openconfig.net/documentation/semantic-versioning/>.

[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 Models", draft-rtgyangdt-rtgwg-device-model-05 (work in progress), August 2016.

[I-D.ietf-netmod-yang-model-classification]
Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", draft-ietf-netmod-yang-model-classification-04 (work in progress), October 2016.

Appendix A. Change summary

A.1. Changes between revisions -01 and -02

- Included additional explanation for release and feature bundles
- Changed feature bundles to be based on schema paths
- Included version 0.2.0 of catalog modules.

A.2. Changes between revisions -00 and -01

- Added release bundle definitions.
- Added IETF module classification identities based on draft-ietf-netmod-yang-model-classification.

Authors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Rob Shakir
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: rjs@rob.sh

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2016

J. Schoenwaelder, Ed.
Jacobs University

June 10, 2016

A Revised Conceptual Model for YANG Datastores
draft-schoenw-netmod-revised-datastores-01

Abstract

Datastores are a fundamental concept binding the YANG data modeling language to protocols transporting data defined in YANG data models, such as NETCONF or RESTCONF. This document defines a revised conceptual model of datastores based on the experience gained with the initial simpler model and addressing requirements that were not well supported in the initial model.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 12, 2016.

Copyright Notice

Table of Contents

1. Introduction

This document provides a revised architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [I-D.ietf-netconf-restconf] and the YANG [RFC6020] data modeling language. Datastores are a fundamental concept binding management data models to network management protocols and agreement on a common architectural model of datastores ensures that data models can be written in a network management protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. Furthermore, the architecture does not detail how data is encoded by network management protocols.

2. Background

NETCONF [RFC6241] provides the following definitions:

- o datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.

- o configuration datastore: The datastore holding the complete set of configuration data that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [I-D.ietf-netmod-rfc6020bis] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG did exist):

- o datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

- o configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration data.

RFC 6244 defined operational state data as follows:

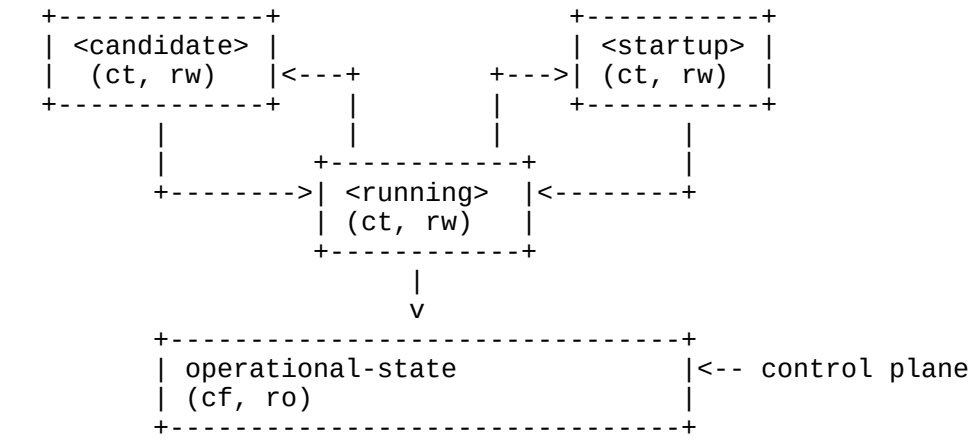
- o Operational state data is a set of data that has been obtained by the system at runtime and influences the system’s behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

Section 4.3.3 of RFC 6244 discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as a separate data tree is the recommended approach.

Implementation experience and requests from operators indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed. Furthermore, separating operational state data from configuration data in a separate branch has been found operationally complicated. The relationship between the branches is not machine readable and filter expressions operating on configuration data and on related operational state data are different.

3. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:



ct = config true; cf = config false; rw = read-write; ro = read-only

Note that read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for the <candidate> and <startup> datastores is optional and the <running> datastore does not have to be writable. Furthermore, the <startup> datastore can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through a <candidate> datastore or by directly modifying the <running> datastore or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to the <running> datastore. NETCONF explicitly mentions so called named datastores.

Some observations:

- Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.

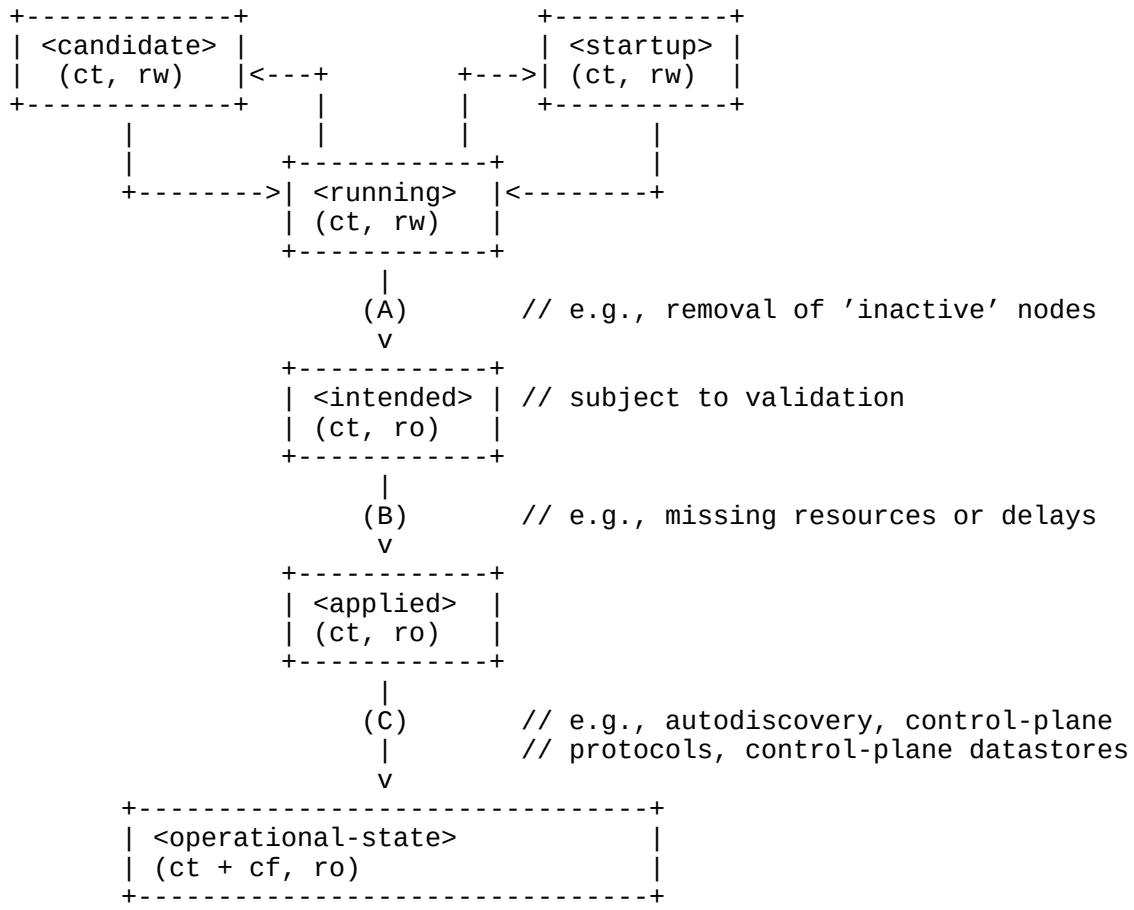
- The NETCONF <get/> operation returns the content of the <running/> configuration datastore together with the operational state. It is therefore necessary that config false data is in a different branch than the config true data. This is in particular relevant if operational state data can have a different lifetime compared to configuration data or if configuration data is not immediately or successfully applied.

- Several implementations have proprietary mechanisms that allow clients to store inactive data in the <running> datastore; this inactive data is only exposed to clients that indicate that they support the concept of inactive data; clients not indicating support for inactive data receive the content of the <running> datastore with the inactive data removed. Validation always happens on the <running> datastore with inactive data removed.

- Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

4. Revised Model of Datastores

Below is a new conceptual model of datastores extending the original model in order reflect the experience gained with the original model.



ct = config true; cf = config false; rw = read-write; ro = read-only

The main changes are:

- The original <running> configuration datastore has been split into the <running> configuration datastore and the <intended> configuration datastore. The <intended> configuration datastore contains the configuration that is intended to be applied to the device. On a traditional NETCONF implementation, <running> and <intended> are always the same. However, implementations that support inactive configuration usually expose <running> to clients that understand inactive configuration and they expose <intended> to clients that do not understand inactive configuration. The introduction of an <intended> datastore makes this difference explicit.

- A new <applied> configuration datastore has been introduced that reflects the configuration currently active on the device. This may be a subset or a superset of the <intended> configuration. Possible reasons for differences are situations where intended configuration can't be applied due to missing resources or where configuration changes take noticeable time to become applied.

- The operational state is considered to reside in a conceptual <operational-state> datastore. This new read-only datastore consists of config true and config false nodes; in the original model the operational state only had config false nodes. The reason for incorporating config true nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

- The model foresees control-plane datastores that are by definition not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The control-plane datastores interact with the rest of the system like any other control-plane mechanisms (e.g., routing protocols, discovery protocols). Note that the ephemeral datastore discussed in I2RS documents maps to a control-plane datastore in the revised datastore model described here.

5. Discussion

5.1. Missing Resources

Sometimes some parts of <intended> configuration refer to resources that are not present and hence parts of the <intended> configuration cannot be applied. A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <applied>.

5.2. System-controlled Resources

Sometimes resources are controlled by the device and such system controlled resources appear in (and disappear from) the operational state dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <applied> eventually (if application of the configuration was successful).

5.3. Auto-configured or Auto-negotiated Values

Sometimes configuration leafs support special values that instruct the system to automatically configure a value. An example is an MTU that is configured to 'auto' to let the system determine a suitable MTU value. Another example is Ethernet auto-negotiation of link speed. In such a situation, the <intended> and <applied> configuration datastores report the value 'auto' while the corresponding leaf in the operational state datastore will report the actual MTU value or the auto-negotiated link speed.

Since a config true leaf may be used both for configuration and for reporting operational state, the value set of a leaf allowed in a configuration datastore may be different from the value set of the corresponding leaf in the operational state datastore.

5.4. Operational State with Different Origins

Sometimes a single list is used to report operational state values that originate from difference sources, i.e., configuration, control plane protocols, or internal processing. An example are IP addresses of an interface that can originate from configuration, from DHCP, or may be dynamically auto-configured. In this case, the operational state datastore will report all IP addresses that are assigned to an interface while the applied configuration datastore only lists the successfully configured addresses that have originated from the intended configuration datastore.

6. Implications

6.1. Implications on NETCONF

- A mechanism is needed to announce support for <intended> and <applied> configuration datastores.
- Support for <intended> and <applied> datastores should be a feature (optional to implement).
- For systems supporting <intended> or <applied> configuration datastores, the <get-config/> operation may be used to retrieve data stored in these new datastores.
- A new operation should be added to retrieve the operational state data store (e.g., <get-state/>). (In principle <get-config/> could work but it would be a confusing name.)
- The <get/> operation will be deprecated since it returns data from two datastores that may overlap in the revised datastore model.

- Invoking <get-config/> on <running> will return <intended> for backwards compatibility. [XXX: How to deal with <edit-config/> for old and new clients with inactive nodes? XXX]

6.2. Implications on RESTCONF

- The {+restconf}/data resource represents the combined configuration and state data resources that can be accessed by a client. This is effectively bundling <running> together with <operational-state>, much like the <get/> operation of NETCONF. The RESTCONF design should change such that the {+restconf}/data resource does not return the content of multiple datastores. Instead, it should return the <running> datastore by default.

- The "content" query parameter can be used to select whether config, nonconfig or all data is returned. It defaults to all. The "content" query parameter should be changed to allow the selection of other datastores, e.g., <operational-state>.

6.3. Implications on YANG

- Some clarifications may be needed if this revised model is adopted. YANG currently describes validation in terms of the <running> configuration datastore while it really happens on the <intended> configuration datastore.

6.4. Implications on Data Models

- Since the NETCONF <get/> operation returns the content of the <running/> configuration datastore and the operational state together in one tree, data models were often forced to branch at the top-level into a config true branch and a structurally similar config-false branch that replicated some of the config true nodes and added state nodes. With the revised datastore model, this is not needed anymore since the different datastores handle the different lifetimes of data objects and together with the deprecation of the <get/> operation it is not possible to write simpler models.

- There may be some differences in the value set of some objects that are used for both configuration and state. At this point of time, these are considered to be rare cases that can be dealt with using text in description statements. Future versions of YANG may consider whether it is reasonable to allow value sets of schema nodes to be partitioned into config true and config false value sets.

- It is desirable to be able to obtain information why a certain value exists in the operational state datastore. Metadata annotations [I-D.ietf-netmod-yang-metadata] should be defined that allow to report the origin of data in the operational state datastore. Note that the definition needs to be flexible and incrementally deployable since not all systems today maintain information about the origin with the operational state.

7. IANA Considerations

None.

8. Security Considerations

This document discusses a conceptual model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

9. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.wilton-netmod-opstate-yang], [I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- Andy Biermann, YumaWorks, <andy@yumaworks.com>
- Martin Bjorklund, Tail-f Systems, <mbj@tail-f.com>
- Marcus Hines, Google, <hines@google.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Anees Shaikh, Google, <aashaikh@google.com>
- Rob Shakir, Jive Communications, <rjs@rob.sh>
- Kent Watsen, Juniper Networks, <kwatsen@juniper.net>

o Robert Wilton, Cisco Systems, <rwilton@cisco.com>

10. References

10.1. Normative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-13 (work in progress), April 2016.

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-13 (work in progress), June 2016.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-05 (work in progress), March 2016.

[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>.

10.2. Informative References

[I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[I-D.wilton-netmod-opstate-yang]
Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2017

A. Sharma
R. Rao
Infinera Corp
X. Zhang
Huawei Technologies

July 6, 2016

Fault YANG Model
draft-sharma-netmod-fault-model-00

Abstract

This document describes the Fault YANG data model for modeling and reporting standing alarm conditions.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 7, 2017.

Table of Contents

1. Introduction

Network devices, controllers, orchestrators, and applications generate faults indicating active alarm state on entities. These faults are reported to the northbound systems, which can diagnose and take corrective actions to fix these faults.

Faults raised by various entities in the system are present in the fault model in the operational datastore. New faults are reported to the clients using notifications. On system re-start, faults are rediscovered, and the fault model is re-populated in the operational datastore. In some systems, faults once cleared, may get moved to historical fault log, which is outside the scope of this document.

The fault definition is based on existing standards [X.733] [RFC3877] and is widely adopted for alarm reporting. This document provides the YANG model for Faults described in the existing standards [X.733] [RFC3877].

2. Fault YANG Data Model

Note: The Fault YANG Data Model contains the most widely used attributes from [X.733]. It is being discussed if this YANG model should contain all attributes as defined in [X.733] and [RFC3877].

The ietf-fault-types YANG model does not currently include all probable causes as defined in X.733. The full list of probable causes will be added in the next version of the document.

New network architectures that include controllers, orchestrators, PCE, applications, etc., require new fault types and probable causes to be defined. These new fault types and probable causes will be defined in the next version of the model.

2.1. YANG Tree

```
module: ietf-fault
   +--ro faults
      +--ro fault* [fault-id]
         +--ro fault-id             inet:uri
         +--ro (entity-type)?
         |  +--:(id)
         |  |  +--ro entity-id           inet:uri
         |  +--:(name)
         |     +--ro entity-name         string
         +--ro fault-type           identityref
         +--ro probable-cause       identityref
         +--ro fault-time           yang:date-and-time
         +--ro fault-severity       enumeration
         +--ro service-affecting?   boolean
         +--ro additional-text?     string
notifications:
   +---n fault-event
      +--ro fault-id             inet:uri
      +--ro (entity-type)?
      |  +--:(id)
      |  |  +--ro entity-id           inet:uri
      |  +--:(name)
      |     +--ro entity-name         string
      +--ro fault-type           identityref
      +--ro probable-cause       identityref
      +--ro fault-time           yang:date-and-time
      +--ro fault-severity       enumeration
      +--ro service-affecting?   boolean
      +--ro additional-text?     string
```

2.2. Fault YANG Model

```
<CODE BEGINS> file "ietf-fault@2016-06-22.yang"

module ietf-fault {
    namespace "urn:ietf:params:xml:ns:yang:ietf-fault";
    prefix flt;

    import ietf-inet-types { prefix "inet"; }
    import ietf-yang-types { prefix "yang"; }
    import ietf-fault-types { prefix "flt-types"; }

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

    contact
```

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

            Editor: Anurag Sharma
                    <mailto:AnSharma@infinera.com>

            Editor: Rajan Rao
                    <mailto:rrao@infinera.com>

            Editor: Xian Zhang
                    <mailto:zhang.xian@huawei.com>";

        description
            "Fault YANG Data Model for Network Topology and Services.";

        revision 2016-06-22 {
            description
                    "Initial revision.";
            reference
                    "TBD";
        }

        grouping fault-entity {
            description
            "Reference to the entity for which the fault is reported.";
            choice entity-type {
                description
                "Entity reference type.";
                case id {
                    leaf entity-id {
                        type inet:uri;
                        mandatory true;
                        description
                        "An identifier for the entity on which the fault
                        is raised. This entity can be in the device,
                        domain controllers, element management systems,
                        or northbound orchestrators. ";
                    }
                }
                case name {
                    leaf entity-name {
                        type string;
                        mandatory true;
                        description
                        "Name for the entity on which the fault is
                        raised. This entity can be in the device,
                        domain controllers, element management systems,
                        or northbound orchestrators. ";
```

```
                }
            }
        }
    }

    grouping fault-info-attributes {
        description
                "Fault Info attributes.";
        leaf fault-id {
            type inet:uri;
            mandatory true;
            description
            "An identifier for the fault. This identifier should be
            chosen such that same fault will always be identified
            using the same identifier.";
        }

        uses fault-entity;

        leaf fault-type {
            type identityref {
                base flt-types:fault-type;
            }
            mandatory true;
            description
            "This parameter categorizes the fault.";
        }

        leaf probable-cause {
            type identityref {
                base flt-types:probable-cause-type;
            }
            mandatory true;
            description
            "This parameter defines further qualification as to the
            probable cause of the alarm.";
        }

        leaf fault-time {
            type yang:date-and-time;
            mandatory true;
            description
            "Time that the fault was raised / reported.";
        }

        leaf fault-severity {
            type enumeration {
                enum Critical {
```

```
                    description
                    "The Critical severity level indicates that a
                    service affecting condition has occurred and an
                    immediate corrective action is required.";
                }
                enum Major {
                        description
                        "The Major severity level indicates that a
                        service affecting condition has developed and
                        an urgent corrective action is required.";
                }
                enum Minor {
                        description
                        "The Minor severity level indicates the
                        existence of a non-service affecting fault
                        condition and that corrective action should
                        be taken in order to prevent a more serious
                        (for example, service affecting) fault.";
                }
                enum Warning {
                        description
                        "The Warning severity level indicates the
                        detection of a potential or impending service
                        affecting fault, before any significant
                        effects have been felt.";
                }
                enum Cleared {
                        description
                        "The Cleared severity level indicates the
                        clearing of one or more previously reported
                        alarms.";
                }
                enum Indeterminate {
                        description
                        "The Indeterminate severity level indicates
                        that the severity level cannot be
                        determined.";
                }
            }
            mandatory true;
            description
            "This parameter indicates the perceived severity level of
            the fault.";
        }

        leaf service-affecting {
            type boolean;
            description
```

```
                "This parameter indicates if the fault impacts an active
                service. If the fault is service affecting then the value
                is true. If the fault does not affect the service then
                the value is false.";
            }

            leaf additional-text {
                type string;
                description
                "This parameter, when present, allows a free form text
                description to be reported.";
            }
        }

        container faults {
            description
            "Serves as top-level container for list of faults.";
            config "false";
            list fault {
                key "fault-id";
                uses fault-info-attributes;
                description
                    "Describes a fault.";
                reference
                            "ITU Recommendation X.733";
            }
        }

        notification fault-event {
            description
            "Fault Notification.";
            uses fault-info-attributes;
        }
    }


    <CODE ENDS>
```

2.3. Fault Types YANG Model

```
<CODE BEGINS> file "ietf-fault-types@2016-06-22.yang"

module ietf-fault-types {
    yang-version 1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fault-types";
    prefix flt-types;

    organization
```

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

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

        Editor: Anurag Sharma
                <mailto:AnSharma@infinera.com>

        Editor: Rajan Rao
                <mailto:rrao@infinera.com>

        Editor: Xian Zhang
                <mailto:zhang.xian@huawei.com>";

    description
        "This module contains Fault data type definitions.";

    revision 2016-06-22 {
        description
                "Initial revision.";
        reference
                "TBD";
    }

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

    identity LOS {
        base probable-cause-type;
        description
                "Loss of Signal.";
    }

    identity LOF {
        base probable-cause-type;
        description
                "Loss of Frame.";
    }

    identity framing-error {
        base probable-cause-type;
        description
                "Framing error probable fault cause.";
    }
```

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

    identity communication-fault-type {
        base fault-type;
        description
                "A fault of this type is principally associated with the
                procedures and/or processes required to convey
                information from one point to another.";
        reference "ITU Recommendation X.733";
    }

    identity QoS-fault-type {
        base fault-type;
        description
                "A fault of this type is principally associated with a
                degradation in the quality of a service.";
        reference "ITU Recommendation X.733";
    }

    identity processing-fault-type {
        base fault-type;
        description
                "A fault of this type is principally associated with a
                software or processing fault.";
        reference "ITU Recommendation X.733";
    }

    identity equipment-fault-type {
        base fault-type;
        description
                "A fault of this type is principally associated with an
                equipment fault.";
    }

    identity environmental-fault-type {
        base fault-type;
        description
                "A fault of this type is principally associated with a
                condition relating to an enclosure in which the
                equipment resides.";
    }
}

<CODE ENDS>
```

3. Security Considerations

TBD

4. IANA Considerations

TBD

5. Acknowledgements

6. Normative References

[RFC3877] Chisholm, S. and D. Romascanu, "Alarm Management Information Base (MIB)", RFC 3877, DOI 10.17487/RFC3877, September 2004, <http://www.rfc-editor.org/info/rfc3877>.

[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>.

[X.733] ITU, ""ITU Recommendation X.733, Information Technology - Open Systems Interconnection - System Management: Alarm Reporting Function"", 1992.

Authors' Addresses

Anurag Sharma
Infinera Corp
169 Java Drive
Sunnyvale, CA 94089
USA

Phone: +1-408-572-5365
Email: AnSharma@infinera.com

Rajan Rao
Infinera Corp
169 Java Drive
Sunnyvale, CA 94089
USA

Phone: +1-408-543-7755
Email: rrao@infinera.com

Xian Zhang
Huawei Technologies
F3-5-B R&D Center, Huawei Industrial Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P.R.China

Email: zhang.xian@huawei.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2017

R. Wilton, Ed.
Cisco Systems

July 6, 2016

Refined YANG datastores with Meta-data
draft-wilton-netconf-opstate-metadata-00

Abstract

draft-wilton-netmod-refined-datastores defines refined YANG datastore definitions to provide an explicit Operational State Datastore and a clean separation between the 'intended configuration' for a device and the 'applied configuration' that is in effect on a device. This draft builds on draft-wilton-netmod-refined-datastores by describing a YANG Metadata based extension that can be used by protocols such as NETCONF and RESTCONF to allow the key information from the various operational state related datatstores to be made available without requiring the client to explicitly read and monitor each abstract datastore separately.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document describes an optional extension to NETCONF/RESTCONF, based on With-defaults Capability for NETCONF [RFC6243], and Metadata with YANG [I-D.ietf-netmod-yang-metadata], that allow servers to communicate the content of the Applied Configuration Abstract Datastore and Operational State Datastore defined in

[I-D.wilton-netmod-refined-datastores] to clients in an efficient way.

Operator requirements state that there needs to be a clear separation between the configuration that has been sent to the device and the actual configuration that the device is actually acting on.

For some simple 'opstate unware' devices, if it is sufficient to assume that the configuration is applied instantaneously and also that none of the configuration can fail, then the intended and applied configuration can be regarded as being exactly the same, and a formal split between intended and applied configuration is unnecessary.

For other more complex 'opstate aware' devices, the assumptions above do not always hold: Devices may take several minutes (or even tens of minutes) to apply the configuration, some of the configuration may fail, or it may not be possible to apply some of the configuration due to either insufficient resources or due to a requisite piece of hardware not being present in the device.

This extension is primarily aimed at this latter class of opstate aware devices, but to improve interoperability it is anticipated that it could also fairly easily be implemented on opstate unaware devices as well.

The extension uses YANG Metadata to allow a client to quickly determine whether the configuration has or has not been successfully applied. Semantically, this is similar to performing a "diff" between two related configuration datastores, but with the content of the diff returned relative to one of those datastores.

1.1. Definitions

Definitions of the following terms are taken from [I-D.wilton-netmod-refined-datastores]:

opstate aware device - a device that implements the requirements specified in [I-D.ietf-netmod-opstate-reqs], in particular the device must expose the split between the device's 'intended configuration' vs 'applied configuration'.

opstate unaware device - a device that is not an 'opstate aware device'. In particular, it does not draw a clear distinction between 'intended configuration' vs 'applied configuration', and generally treats them as having exactly the same contents.

abstract datastore - a new variant of YANG datastore that represents a particular common property of the contents (e.g. 'applied configuration'). Servers could allow it to be external referenced as a named datastore, but generally that is not expected or required.

The following datastore definitions are taken from NETCONF [RFC6241]:

- candidate ds - represents candidate configuration
- startup ds - represents startup configuration

The following datastore definitions are taken from [I-D.wilton-netmod-refined-datastores]:

- Persistent Configuration - holds the client provided configuration that is written to the Startup Datastore and is recovered after device reboot.
- Ephemeral Configuration - an optional datastore that holds client provided transient configuration that is discarded after device reboot.
- Operational State - a read-only datastore that holds all of the operational state of the device. Specifically it holds: the exact configuration that has been applied, along with any system controlled configuration, and all system state (including statistics and any ephemeral state).
- Intended Configuration - abstractly represents the combined desired configuration of the device
- Applied Configuration - abstractly represents the actual applied configuration of the device
- Running Configuration - abstractly represents the combined intended and applied datastores, logically equivalent to the definition given in [RFC6241]

1.2. Objectives

The objectives of this draft are:

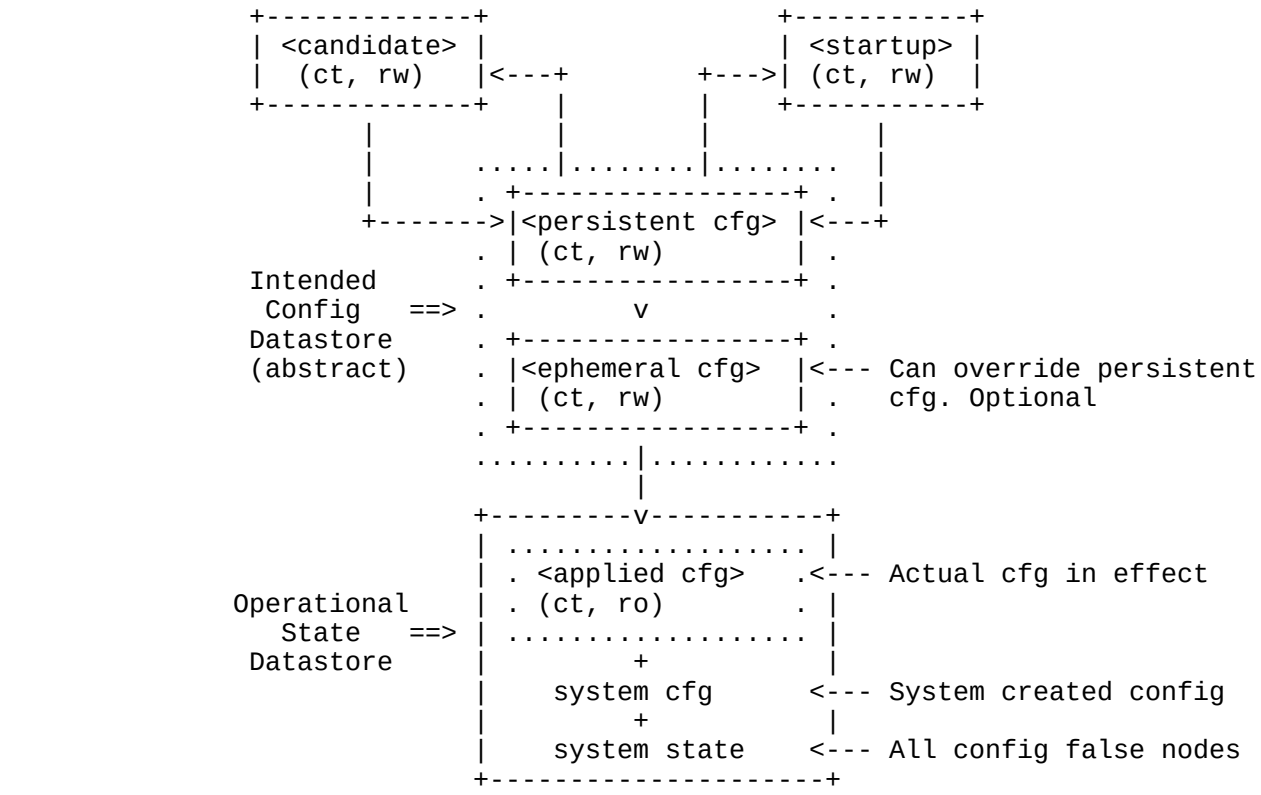
to minimize the number of explicit datastores that NETCONF/ RESTCONF servers must implement and clients must manage, whilst providing a clean separation between intended configuration, applied configuration, and operational state.

to provide an efficient mechanism to query and monitor whether all of the intended configuration has been applied, and view any configuration that has not been applied.

to provide an efficient mechanism to query and monitor the operational state of the device.

1.3. Overview of a refined model of datastores

The following diagram, taken from [I-D.wilton-netmod-refined-datastores], illustrates how all of the abstract and concrete datastores (except running) relate to each other:



Key
Solid boxes (-----) indicate normal datastores:
(i.e Startup, Persistent Cfg, Ephemeral Cfg, Operational State)

Dotted boxes (.....) indicate abstract datastores:
(i.e. Intended Config and Applied Config)

ct = config true, rw = read/write, ro = read/only

2. Applied Configuration Metadata Option to NETCONF and RESTCONF

The applied configuration metadata option is an optional extension to NETCONF/RESTCONF, loosely based on With-defaults Capability for NETCONF [RFC6243], that uses YANG Metadata [I-D.ietf-netmod-yang-metadata] to annotate the Persistent Configuration, Ephemeral Configuration, Operational State, or Running Configuration Datastores with additional metadata that partially reflects the contents of the abstract datastores illustrated in Section 1.3.

Principally, the metadata annotations allow the client to see whether the configuration has been applied or failed; the reason for the failure if it has failed; or whether the configuration node only exists because it has been created by the system. This is achieved without the client having to query or explicitly monitor any of the extra abstract datastores.

2.1. 'with-applied-config-metadata' modes query parameter

The 'with-applied-config-metadata' option supports further refinement of the nodes and metadata that is returned through a query parameter that supports three modes.

2.1.1. selectively-annotate

For a given NETCONF/RESTCONF query, the 'selectively-annotate' option returns all the nodes what would have been returned for that query anyway, but with additional metadata annotations for any nodes in the request datastore that differ from the equivalent node in the applied configuration datastore (i.e. it would exclude metadata annotations for nodes that would be reported as "cfg-status=applied").

This is the default option and MUST be supported by all servers implementing the with-applied-config option.

This option allows a client to both monitor the exact configuration that the device is trying to converge to, and also the status of whether that configuration has been applied. This is achieved in a way that uses a single request, and in the context of a single datastore. In the case that the applied configuration exactly matches the intended configuration then clearly no additional metadata annotations are returned.

2.1.2. annotate-all

For a given NETCONF/RESTCONF query, the 'annotate-all' option returns all the nodes what would have been returned for that query anyway, but with opstate metadata annotations for all nodes that are returned. I.e. this is the same data that would be returned using the 'selectively-annotate' option except that it also includes metadata for nodes that are reported as "cfg-status=applied".

This option MAY be supported by servers implementing the opstate metadata option. It may be useful for clients that prefer explicit applied configuration annotations for every node.

2.1.3. annotated-diff

For a given NETCONF/RESTCONF query, the 'annotated-diff' option filters the nodes returned in the standard query, so that only nodes that are not reported as "cfg-status=applied" are returned. Applied configuration metadata annotations are included for all nodes that are returned.

This option makes it easy for clients to determine whether their configuration is fully applied in a concise way and SHOULD be supported by all servers implementing the with-applied-config option.

2.2. Datastore specific handling

The Applied Configuration Metadata could be used with any of following configuration datastores (persistent, ephemeral, intended, applied, running), or the Operational State Datastore. However, not all values of the "cfg-status" make sense for all datastores, so the values that may be used for particular datastores are restricted as follows:

Persistent Cfg Ds - applying, applied, applied-deviation, overridden, blocked, failed

Ephemeral Cfg Ds - applying, applied, applied-deviation, overridden, blocked, failed

Intended Cfg Ds - applying, applied, applied-deviation, blocked, failed

Applied Cfg Ds - applying, applied, applied-deviation, failed

Operational State Ds - applying, applied, applied-deviation, system-controlled, failed

If the Running Configuration Datastore is handled as described in [I-D.wilton-netmod-refined-datastores], then NETCONF <get> requests are handled as if they are against the Operational State Datastore, and <get-config> requests are handled as if they are made against the Intended Configuration Datastore, which indicates which metadata values can be used.

2.3. Applied Configuration Metadata YANG Definition

Defines the applied configuration datastore metadata that is used in both NETCONF and RESTCONF

```
<CODE BEGINS> file "ietf-yang-opstate-metadata@2016-07-06.yang"
module ietf-yang-opstate-metadata {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-opstate-metadata";

  prefix "opstate";

  import ietf-yang-metadata {
    prefix "md";
  }

  organization
    "IETF NETCONF (Network Configuration Protocol) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>

     WG List:  <netconf@ietf.org>

     WG Chair: Mahesh Jethanandani
               <mjethanandani@gmail.com>

     WG Chair: Mehmet Ersue
               <mehmet.ersue@nsn.com>

     Editor:   Robert Wilton
               <rwilton@cisco.com>";

  description
    "This module defines YANG metadata to allow the reason why
     a config true node exists in the operational state datastore
     to be determined using YANG metadata.

     Copyright (c) 2016 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
     draft-wilton-netconf-opstate-metadata-00; see the Internet
     draft itself for full legal notices.";

  revision 2016-07-06 {
```

```
       description "Initial revision";

       reference
         "Internet draft: draft-wilton-netconf-opstate-metadata-00";
     }

     md:annotation cfg-status {
       type enumeration {
         enum applying {
           description
             "The configuration for the annotated node is currently
              changing (i.e. being created, deleted or changing in
              value) as part of an ongoing configuration operation";
         }
         enum applied {
           description
             "The configuration is fully applied.  The node exists in
              both the intended and applied datastores and has exactly
              the same value in both.";
         }
         enum applied-deviation {
           description
             "The configuration has been applied to the extend the
              server is able to, but the value in the applied
              configuration datastore does not exactly match the value
              in the intended configuration datastore.";
         }
         enum overridden {
           description
             "The configuration node value has been overridden by the
              same node in another configuration datastore.";
         }
         enum system-controlled {
           description
             "The configuration node only exists in the Operational
              State Datastore because it is system controlled. It is
              not present in the abstract applied configuration
              datastore.";
         }
         enum blocked {
           description
             "The system cannot apply the configuration because
              the required hardware resources are not present.  The
              configuration node does not exist in the applied
              configuration datastore.";
         }
         enum failed {
           description
```

```
            "The system cannot apply the configuration due to an
             error.  The configuration node does not exist in the
             applied configuration datastore.";
        }
      }
      description
        "Status indicates why a configuration node (i.e. config=true)
         in the operational-state datastore does not match the
         corresponding node in the intended config datastore";
    }

    md:annotation cfg-status-reason {
      when "../status = 'blocked' or ../status = 'failed'" {
        description
          "An optional status reason can be provided for blocked or
           failed configuration";
      }
      type string;
      description
        "Indicates the reason why the applied configuration node is
         blocked or failed";
    }
  }
  <CODE ENDS>
```

2.4. :with-applied-cfg-metadata Capability

The :with-applied-cfg-metadata capability for NETCONF and RESTCONF indicates that the server is capable of returning applied configuration metadata.

2.4.1. Capability Identifier

Equivalent capabilities are defined for both NETCONF and RESTCONF:

NETCONF: urn:ietf:params:netconf:capability:with-applied-cfg-metadata:1.0

RESTCONF: urn:ietf:params:restconf:capability:with-applied-cfg-metadata:1.0

Note that this protocol capability URI is separate from the YANG module capability URI for the YANG module is Section XXX. A server that implements this module MUST also advertise a YANG module capability URI according to the rules specified in [RFC6020].

2.4.1.1. datastore parameter

The identifier MUST have a parameter: "supported-datastores". This indicates which datastores the server allows the :with-applied-cfg-metadata option to be used on.

The datastores that may be supported are described in Section 1.1. The allowed values of this parameter include 'persistent', 'ephemeral', 'opstate', 'running', and any other appropriate configuration datastores supported by the server. Both 'persistent' and 'operational state' datastores MUST be supported.

2.4.1.2. supported-modes parameter

The identifier MUST also have the parameter: "supported-modes". This indicates which particular operations are supported.

Possible modes are 'selectively-annotate', 'annotate-all', and 'annotated-diff' as defined in Section 2.1. The 'selectively-annotate' option MUST be supported, and the 'annotated-diff' option SHOULD be supported.

2.4.1.3. Examples

NETCONF and RESTCONF capability examples:

> urn:ietf:params:netconf:capability:with-applied-cfg-metadata:1.0?supported-datastores=persistent,intended&supported-modes=selectively-annotate
>
> urn:ietf:params:restconf:capability:with-applied-cfg-metadata:1.0?supported-datastores=running&supported-modes=selectively-annotate,annotated-diff

2.5. NETCONF specifics

Further details for the <with-applied-config-metadata> option need to still be fleshed out here:

> The option would only be supported for NETCONF <get> and <get-config> requests.
>
> NETCONF would need to add support for the new datastores (as least "persistent", "opstate").

2.6. RESTCONF specifics

Further details for the <with-applied-config-metadata> option need to still be fleshed out here:

The option would only be supported for RESTCONF GET requests.

The option could be used when accessing the default combined datastore view, or with new datastore specific REST paths (e.g. as least "persistent", "opstate").

3. Discussion Points

This section lists some points that may warrant further discussion:

The proposal is written in terms of NETCONF/RESTCONF "get" requests but it is desirable if the metadata could also apply to YANG Pub/Sub as well.

Should the extension target adding opstate specific metadata, or is it just applied configuration metadata?

The proposed YANG model merges several different opstate properties into a single 'cfg-status' leaf, possibly these could be separated out into separate leaves.

One of the aims of the approach described in [I-D.wilton-netmod-refined-datastores] and in this document is to allow opstate unaware servers to fairly easily add basic support for the operational state extensions. This provides an opportunity to improve interoperability with NETCONF/RESTCONF clients because they can treat opstate aware and opstate aware servers in exactly the same way. Does the approach in this draft achieve this goal?

4. Acknowledgements

This document arose through discussions to find a consensual solution to the operational state problem. The following people were actively involved in the discussions that indirectly led to this document:

- Lou Berger, Labn Consulting, <lberger@labn.net>
- Martin Bjorklund, Tail-f Systems, <mbj@tail-f.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Acee Lindem, Cisco Systems, <acee@cisco.com>

- Juergen Schoenwaelder, Jacobs University Bremen <j.schoenwaelder@jacobs-university.de>

- Rob Shakir, Jive Communications, <rjs@rob.sh>

- Kent Watsen, Juniper Networks, <kwatsen@juniper.net>

The author would also like the thank the following people who have kindly provided feedback on this draft: Matt Hall, Einar Nilsen-Nygaard.

5. IANA Considerations

None

6. Security Considerations

TBD.

7. References

7.1. Normative References

[I-D.ietf-netmod-opstate-reqs] Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[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>.

[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>.

7.2. Informative References

[I-D.ietf-i2rs-ephemeral-state] Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-10 (work in progress), June 2016.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-07 (work in progress), March 2016.

[I-D.schoenw-netmod-revised-datastores]
Schoenwaelder, J., "A Revised Conceptual Model for YANG Datastores", draft-schoenw-netmod-revised-datastores-01 (work in progress), June 2016.

[I-D.wilton-netmod-refined-datastores]
Wilton, R., "Refined YANG datastores", draft-wilton-netmod-refined-datastores-00 (work in progress), June 2016.

[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>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

Appendix A. Usage examples

A sample encoding of the <with-applied-cfg-metadata> enhancement is described below.

A simple example module is provided to illustrate the subsequent examples. This is not a real module, and is not intended for any real use.

```
module example-interfaces {

  namespace "http://example.com/ns/interfaces";

  prefix exam;

  container interfaces {
    description "Example interfaces group";

    list interface {
      key name;
      description "Example interface entry";
```

```
        leaf name {
          type string {
            length "1 .. max";
          }
          description
            "The administrative name of the interface.";
        }

        leaf mtu {
          type uint32;
          default 1514;
          description
            "The maximum transmission unit (MTU) value assigned to
             this interface.";
        }

        leaf enabled {
          type boolean;
          default "true";
          description "Enable the interface";
        }

        leaf oper-status {
          config false;
          type enumeration {
            enum up {
              description
                "Ready to pass packets.";
            }
            enum down; {
              description
                  "The interface does not pass any packets.";
            }
          }
        }
      }
    }
  }
```

A.1. NETCONF Persistent datastore get-config request using with-applied-cfg-metadata

This example illustrates a <get-config> request made against the Persistent Configuration Datastore using the with-applied-cfg-metadata selectively-annotate option using the example YANG module above:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <persistent/>
    </source>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-applied-cfg-metadata>
     xmlns="urn:...:ietf-netconf-with-applied-cfg-metadata">
      selectively-annotate
    </with-applied-cfg-metadata>
  </get-config>
</rpc>
```

The response indicates that at the time of the reply:

> The system has failed to apply the MTU configuration of 9001 on eth0/0 due to a hardware programming error.
>
> The request to change the MTU leaf on eth0/1 to 9000 is still in the process of being applied.
>
> The MTU configuration on eth0/2 has been successfully applied.

```
<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:opstate="urn:...:ietf-yang-opstate-metadata">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <name>eth0/0</name>
        <mtu opstate:cfg-status="failed"
             opstate:cfg-status-reason="hardware programming error">
          9001
        </mtu>
      </interface>
      <interface>
        <name>eth0/1</name>
        <mtu opstate:cfg-status="applying">
          9000
        </mtu>
      </interface>
      <interface>
        <name>eth0/2</name>
        <mtu>2000</mtu>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.2. NETCONF Operational State Datastore get request using with-applied-cfg-metadata

This example illustrates a <get> request made against the Operational State Datastore using the with-applied-cfg-metadata selectively-annotate option using the example YANG module above:

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <source>
      <opstate/>
    </source>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-applied-cfg-metadata>
     xmlns="urn:...:ietf-netconf-with-applied-cfg-metadata">
      selectively-annotate
    </with-applied-cfg-metadata>
  </get>
</rpc>
```

An example response is given below. This response assumes that the device is in the same state as for the <get-config> example given above, and assumes that the device uses NETCONF with-default "explicit" mode. The response indicates that at the time of the reply:

The MTU on eth0/0 is still at the YANG default of 1514. This differs from the intended configuration because the configuration failed to be applied.

The configuration request to change the MTU leaf on eth0/1 from 1514 to 9000 is still in the process of being applied, and the device is still currently using an MTU of 1514.

The MTU configuration of 2000 on eth0/2 has been successfully applied.

eth0/3 has not been configured, but exists in the Operational State Datastore because it is automatically created by the device. The MTU leaf (which has the default value) must also be marked as system-controlled because there is no implicit or explicit MTU leaf in the intended configuration for eth0/3. The enabled leaf is also system-controlled but with a non default value, since the device does not allow the interface to be enabled without being explicitly configured.

```
   <rpc-reply message-id="101"
              xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
              xmlns:opstate="urn:...:ietf-yang-opstate-metadata">
     <data>
       <interfaces xmlns="http://example.com/ns/interfaces">
         <interface>
           <name>eth0/0</name>
           <mtu opstate:cfg-status="failed"
                opstate:cfg-status-reason="hardware programming error">
             1514
           </mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name>eth0/1</name>
           <mtu opstate:cfg-status="applying">
             1514
           </mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name>eth0/2</name>
           <mtu>2000</mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name opstate:cfg-status="system-controlled">
             eth0/3
           </name>
           <mtu opstate:cfg-status="system-controlled">
             1514
           </mtu>
           <enabled opstate:cfg-status="system-controlled">
             false
           </enabled>
           <oper-status>down</oper-status>
         </interface>
       </interfaces>
     </data>
   </rpc-reply>
```

Author's Address

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

R. Wilton, Ed.
Cisco Systems

July 7, 2016

Refined YANG datastores
draft-wilton-netmod-refined-datastores-01

Abstract

The existing definition of YANG datastores supported by NETCONF only provides a loose definition of the running datastore, and no formal definition of any datastore that represents the operational state of a device. This document defines a refined datastore model with new concrete and abstract datastores to allow devices to provide a clean separation between an operator's intended configuration of a device and the actual running operational state of a device. It provides a similiar, but alternative, datastore framework to the one described in draft-schoenw-netmod-revised-datastores.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 8, 2017.

Table of Contents

1. Introduction

This document defines a similar, but alternative, architectural datastore framework to [I-D.schoenw-netmod-revised-datastores]. The aim of this document is to make it easier to compare the models and approaches being described in the two different drafts. The reader is advised to also read [I-D.schoenw-netmod-revised-datastores] which provides a good background on why a refined NETCONF/YANG datastore model is needed.

This document defines serveral new datastores, and also introduces the new concept of an abstract datastore. Despite multiple new datastores being defined, the expectation is that clients and devices would mainly interact with only the Persistent Configuration and Operational State Datastores. Candidate and Startup datastores would be used as presently defined, and the Running datastore would be supported (as much as possible) for backwards compatibility purposes.

Abstract datastores are a new flavor of datastore that are semantically equivalent to regular datastores, but without the expectation that clients would be able to explicitly interact with them as explicitly named datastores. Instead, clients would likely infer the contents of the abstract datastores through metadata annotations on the regular datastores. One of the main advantages for defining these abstract datastores is to allow for a precise definition of the meaning of the configuration and operational data on a device, and potentially allow management agents to make comparisons between the different datastores. E.g. to determine if any intended configuration has not actually been applied, or perhaps conversely which parts of the applied configuration do not match the intended configuration.

This document also gives an idea of how ephemeral state could potentially be represented to meet the I2RS requirements specified in [I-D.ietf-i2rs-ephemeral-state]. Ephemeral configuration is treated as a separate optional configuration datastore that constitutes part of the intended configuration of the device. Ephemeral operational state is represented as part of the Operational State Datastore described in Section 5.8. Further refinement of the proposed handling of ephemeral state is likely needed to ensure that all the I2RS ephemeral state requirements are met.

2. Objectives

The key aims of the datastore definitions given in this document are:

to keep the existing definition of the running-configuration unchanged,

to minimize the impact on existing NETCONF/RESTCONF implementations, and to provide a viable upgrade path for existing NETCONF/RESTCONF servers,

to minimize the number datastores (and amount of data) that need to be explicitly managed by external management agents,

to make explicit the meaning of the contents of each datastores and how they relate to each other.

3. Definitions

The following terms are defined in [I-D.ietf-netmod-opstate-reqs]:

Intended Configuration

Applied Configuration

Operational State

The following definition is taken from [RFC6241]:

running configuration datastore - A configuration datastore holding the complete configuration currently active on the device. The running configuration datastore always exists.

The following new terms are defined here:

opstate aware device - a device that implements the requirements specified in [I-D.ietf-netmod-opstate-reqs], in particular the device must expose the split between the device's 'intended configuration' vs 'applied configuration'.

opstate unaware device - a device that is not an 'opstate aware device'. In particular, it does not draw a clear distinction between 'intended configuration' vs 'applied configuration', and generally treats them as having exactly the same contents.

abstract datastore - a conceptual type of datastore that represents some abstract property (e.g. 'applied configuration') of the data nodes that it contains. Although devices could allow

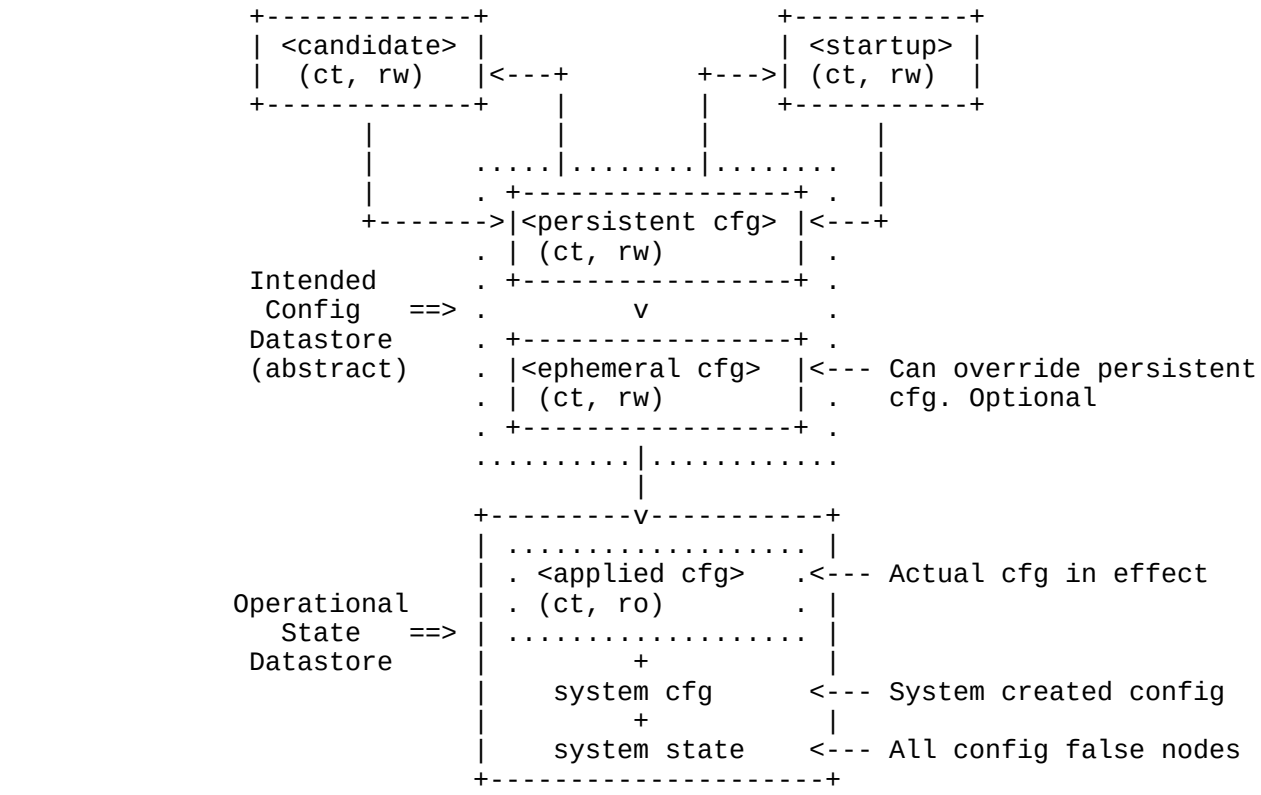
an abstract datastore to be externally referenced as a named datastore, this is not expected or required.

4. Overview of a refined model of datastores

This document defines a refined datstore model that can universally be used for both opstate aware devices and also existing NETCONF/ RESTCONF servers. The model is intended to cover both the opstate requirements [I-D.ietf-netmod-opstate-reqs] and the I2RS ephemeral configuration datastore requirements [I-D.ietf-i2rs-ephemeral-state].

All datastores described in this document use the same YANG schema, although the actual data nodes that are allowed in a particular datastore can depend on statements set in the schema. For example, the configuration datastores only contain datanodes for YANG schema nodes that are "config=true".

The following diagram illustrates how the datastores (except 'Running') relate to each other:



Key
Solid boxes (-----) indicate normal datastores:
(i.e Startup, Persistent Cfg, Ephemeral Cfg, Operational State)

Dotted boxes (.....) indicate abstract datastores:
(i.e. Intended Config and Applied Config)

ct = config true, rw = read/write, ro = read/only

Three new datastores are defined:

- Persistent Configuration - holds the current, client provided, configuration for the device that is saved to the Startup Datastore and is recovered after device reboot.

- Ephemeral Configuration - an optional datastore that holds client provided transient configuration that is discarded after device reboot.

- Operational State - a read-only datastore that holds all of the operational state of the device. Specifically it holds: the exact configuration that has been applied, along with any system controlled configuration, and all system state (including statistics and any ephemeral state).

Two new abstract datastores are defined:

- Intended Configuration - represents the combined desired configuration of the device

- Applied Configuration - represents the actual applied configuration of the device

Two datastores are unchanged from the NETCONF [RFC6241] definition:

- Candidate - represents candidate configuration

- Startup - represents startup configuration

For opstate aware devices, the Running Configuration Datastore is redefined as an abstract datastore that represents the combined persistent and applied configuration. It is regarded as a logical expansion of the definition in NETCONF [RFC6241].

5. Definition of all datastores

5.1. Candidate Datastore (optional)

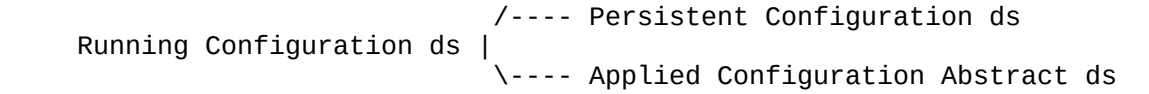
Holds candidate configuration. Unchanged from NETCONF [RFC6241].

5.2. Startup Datastore

Holds the saved configuration that is used by the device after reboot. Unchanged from NETCONF [RFC6241].

5.3. Running Configuration Datastore

To accommodate a clean separation between configuration and state, for an opstate aware device, the Running Configuration Datastore is logically split into two component parts, the Persistent Configuration Datastore and Applied Configuration Datastore, as illustrated by the following diagram:



The Applied Configuration Abstract Datastore is part of the Operational State Datastore.

5.4. Persistent Configuration Datastore

The Persistent Configuration Datastore holds the current configuration provided by a client that is expected to be persisted over device reboot. The datastore can be read and written by a client. Any edit operations on the datastore are validated as per YANG constraints validation before being processed further. The persistent configuration datastore interacts with both the Candidate and Startup datastores, and forms part of the Intended Configuration Abstract Datastore.

5.5. Ephemeral Configuration Datastore (Optional)

The Ephemeral Configuration Datastore may optionally be supported to hold any configuration that must not persist over device reboot. This writable datastore could be regarded as a pane of glass overlay on the persistent configuration datastore, allowing nodes in the ephemeral configuration to override, or depend on, nodes in the persistent configuration if required.

A mechanism is required to allow a client to choose which values take precendence if a leaf with different values exists in both the persistent configuration and ephemeral configuration datastore.

Multiple layers of ephemeral configuration within the ephemeral datastore could be supported.

5.6. Intended Configuration Abstract Datastore

The Intended Configuration Abstract Datastore represents the entire combined intended configuration for the device. It is implemented as the net combination of the Persistent Configuration Datastore combined with the optional Ephemeral Configuration Datastore.

For devices that do not support ephemeral configuration, the Intended Configuration Abstract Datastore is exactly the same as the Persistent Configuration Datastore.

5.7. Applied Configuration Abstract Datastore

The Applied Configuration Abstract Datastore is the subset of the config true datanodes in the Operational State Datastore that represents the applied configuration on the device.

If the intended configuration has been completely successfully applied, then the contents of the Applied Configuration Abstract Datastore exactly matches the Intended Configuration Abstract Datatstore, conforming to the behaviour specified in [I-D.ietf-netmod-opstate-reqs].

5.8. Operational State Datastore

The Operational State Datastore represents all of the operational state of the device, and is made up of the applied configuration, along with any system controlled configuration, and all of the system state. It includes statistics and ephemeral state (both applied configuration and operational state nodes).

This datastore contains all nodes defined in the YANG schema (i.e. both config true and config false nodes). The contents of this datastore can only be updated by the device, and as such, from a client perspective this datastore is read only.

The data held in the Operational State Datastore can be further classified into five categories:

5.8.1. Applied Configuration

The Operational State Datastore contains the applied configuration that represents the configuration that the device is actual using to operate the device.

If the intended configuration has been completely successfully applied, then the applied configuration data nodes exactly matches the logical contents of the Intended Configuration Abstract Datatstore.

5.8.2. System Controlled Configuration

In addition to the applied configuration, the Operational State Datastore also contains any System Controlled Configuration data nodes. These data nodes, using the same YANG config true schema nodes as for the applied configuration, represent all configuration that is created and controlled by the system independently of the applied configuration. E.g. this would include system created interfaces, which exist in the Operational State Datastore regardless of whether they have been explicitly configured by a client.

There is no YANG schema keyword required to identify nodes that may be system controlled. Hence, a device could choose for any config true node in the YANG schema to be system controlled, but a device would be expected to identify to a client the data nodes in the

Operational State Datastore that are system controlled through a mechanism such as YANG Metadata (e.g. as described in [I-D.wilton-netconf-opstate-metadata]).

5.8.3. System State

System State represents all of the data nodes in the Operational State Datastore that are represented by config false nodes in the YANG schema.

5.8.4. Statistics

Statistics are a subset of the data nodes in the Operational State Datastore. Statistics nodes are identified in the YANG schema by the "statistics true" statement, that must also be identified as "config false".

5.8.5. Ephemeral State

Ephemeral state is the subset of the schema in the Operational State Datastore that represents ephemeral state nodes, which are identified in the YANG schema by the ephemeral keyword, including both the applied ephemeral configuration (config true, ephemeral true), and ephemeral operational state (config false, ephemeral true).

6. Discussion points

6.1. A complete and accurate representation of a device’s configuration

Sometimes a device cannot completely implement all of the nodes and values specified by a YANG schema. Ideally a well behaved device would indicate which parts of the schema it cannot completely support via YANG deviations. But deviations do not work in all scenarios - support for particular values of a configuration leaf may be dependent on the underlying hardware that is present in the device at the time. In this and other similar scenarios, to ensure that a client can properly manage a device, the applied configuration must be a complete and accurate representation of all of the configuration that a device is actually running. This must include any features that are implicitly enabled by default without any client configuration, or places where the applied configuration value differs from the intended configuration value (e.g. perhaps to protect the system from being overloaded).

6.2. Missing resources

Configuration that cannot be applied because the system resources are missing (or have been exhausted) would logically exist in the intended configuration datastore but not in the applied configuration datastore.

As defined in [I-D.ietf-netmod-opstate-reqs], by default, a NETCONF or RESTCONF configuration commit would be expected to fail if some of the configuration was for system resources that were not present. Extensions to NETCONF and RESTCONF could be provided to allow for more control over configuration operations that contain configuration for system resources that are missing.

6.3. System controlled resources

System controlled resources (i.e. those resources that would exist in the system regardless of configuration) always exist as nodes in the Operational State Datastore as part of the "system controlled configuration". If the resources have also been configured then they would also be present in the abstract intended datastore, and if the configuration successfully applied, the abstract applied configuration datastore as well.

Clients could find out which nodes in the operational state datastore are system controlled by using YANG Metadata, e.g. as described in [I-D.wilton-netconf-opstate-metadata].

6.4. Auto-configured or auto-negotiated values

The applied configuration in the Operational State Datastore only represents the configuration that is applied, and is bound by the constraint that if the configuration has been successfully applied then it must exactly match the intended configuration. Hence, this generally requires that separate config false leaves in the Operational State Datastore are required to represent the exact values programmed in the device.

In the specific case that the operational value meets the following three constraints then no separate config false leaf is required:

1. it is directly controlled by configuration,

2. it has exactly the same schema value space as the corresponding configuration leaf, and

3. if configured, the operational value of the system matches the applied configuration.

This optimization is allowed because the config false leaf value in the Operational State Datastore would always have exactly the same lifecycle existence and value as the corresponding config true leaf representing the applied configuration in the Operational State Datastore.

6.5. Operational State with Different Origins

The definition of the Operational State Datastore is designed to allow config false leaves to depend on either explicitly configured entities (in the applied configuration datastore) or system created configuration entries. This definition facilitates the merging of separate configuration and state parts of YANG models into the same container/lists if desired.

An example are IP addresses of an interface that can originate from configuration, from DHCP, or may be dynamically auto-configured. In this case, the operational state datastore would contain all IP addresses. The explicitly configured addresses would logically exist in the abstract applied configuration datastore. Addresses learned through DHCP or dynamically configured would logically exist as system controlled config true data nodes in the Operational State Datastore. Enhanced get/notification requests with YANG metadata annotations could be used to amend the reply/notification with metadata information to indicate which datastore the entries logically exist in.

6.6. Statistics

Both the Overview of 2002 IAB Network Management Workshop [RFC3535] and NETCONF/YANG Network Management Architecture [RFC6244] categorises the state of a devices separately into configuration, operational state, and statistics. NETCONF and YANG have historically only categorised the state of a device into configuration and non-configuration.

This document considers statistics to be part of the Operational State Datastore, which is consistent with how statistics information is returned in current NETCONF/RESTCONF requests, and allows all operational state to be efficiently and easily retrieved together in one request. It is envisaged that YANG schema data nodes could be labelled with a new "statistics true" statement to allow for easy filtering of statistics data for NETCONF/RESTCONF GET/GET-CONFIG requests and also YANG pub/sub. I.e. the extensions should allow for requests against the Operational State Datastore that exclude statistics, along with requests that only include statistics (along with any necessary containing structure and keys).

6.7. YANG Defaults Handling

The protocol handling of YANG defaults is described in NETCONF With-defaults [RFC3535] and RESTCONF [I-D.ietf-netconf-restconf]. These documents allow a device to report how YANG defaults are normally handled in requests for data resources, but with the expectation that the same semantics apply to all datastores. There is no way to express that the default handling semantics should depend on the datastore that a request pertains to.

When considering operational state, the most useful semantics for handling YANG defaults depend on the particular datastore and what system data it represents. To allow servers to provide optimal default handling, it is proposed that an extension to, or new version of, With-defaults is defined to support the proposed semantics below:

For configuration datastores that directly represent client provided configuration (i.e. the Persistent Configuration Datastore, Ephemeral Configuration Datastore and Intended Configuration Abstract Datastore), the most useful semantics are to return the exact data nodes set by the client (i.e. this is equivalent to the With-defaults "explicit" capability mode). Using this mode makes it easy for clients to check that a device has received and is acting on the exact desired configuration. Further, clients can always strip default values out of the configuration that they send to the device if they wish.

For the Operational State Datastore, which represents the exact operational state of the device, it is most helpful if the device returns the exact operational state of the device including any data nodes with a value that matches the YANG schema default value (i.e. this is equivalent to the With-defaults "report-all" capability mode). The benefits to the client of being able to rely on the values provided by the device outweigh the slight increase in data and processing overhead.

In addition, it is desirable if that the new with-defaults semantics apply to both explicit NETCONF/RESTCONF GET/GET-CONFIG requests and also YANG pub/sub. In all cases, for servers that support the YANG With-defaults extension, clients can overide the default handling behavior to whichever semantics they desire.

7. Implications of the Refined Datastore Model

7.1. Implications for YANG

The proposed revised datastore definitions have the following implications on YANG:

- A new "statistics true|false" statement is added to the YANG language to label schema data nodes that only contain statistics:
  - Only "config false" schema data nodes may be labelled "statistics true".
  - Schema data nodes that are labelled "statistics true" may not have any decendent child nodes that are either labelled "config true" or "statistics false".
- A new "ephemeral true|false" statement is added to the YANG language to label schema data nodes that contain ephemeral state:
  - Schema data nodes labelled "config true" or "config false" may also be labelled "ephemeral true".
  - Schema data nodes labelled "statistics true" or "statistics false" may also be labelled "ephemeral true".
  - Schema data nodes labelled "config true" and "ephemeral true" cannot appear in the Persistent Configuration Datastore.
  - Schema data nodes that are labelled "ephemeral true" cannot have any decendent YANG schema nodes that are labelled "ephemeral false".

7.2. Implications for NETCONF

The proposed revised datastore definitions have the following implications on NETCONF:

- Support for the new configuration datastores is added to NETCONF:
  - <get> and <get-config> requests are supported on the new datastores, but both return the same data.
  - Only the Persistent Configuration and Ephemeral Configuration (along with Candidate/Startup) datastores are writable by clients.
  - It is an implementation choice whether devices support Intended Config and Applied Config as named, client accessable, datastores.

    * A new NETCONF capability would indicate which new configuration datastores are supported by the device.

  o Support for the new Operational State Datastore is added to NETCONF:

    * <get> requests return all the data from the datastore.

    * <get-config> requests only return config true data nodes (applied config and system controlled config) from the datastore.

    * <edit-config> requests are not supported.

    * A new NETCONF capability would indicate that the device supports the Operational State Datastore.

7.2.1. Implications for Opstate Aware Devices

The following implications are specific to opstate aware devices supporting NETCONF:

  o Configuration requests on the Running Configuration Datastore are handled as follows:

    * <edit-config> and <get-config> requests are mapped to the Persistent Configuration Datastore.

    * <get> requests are mapped to the Operational State Datastore.

7.2.2. Implications for Opstate Unaware Devices

One of the key aims of the refined datastore model described in this draft is to minimize the impact on existing (or opstate unaware) NETCONF/RESTCONF clients and devices. The assumption of this model is that for an opstate unaware device, the Persistent Configuration, Intended Configuration and Applied Configuration Datastores all hold exactly the same values, and are collectively labelled as the Running Configuration Abstract Datastore).

An opstate unaware device does not have to make any changes, but a device could add support for the following to maximize interoperability:

  o All config requests on the Persistent Configuration, Intended Configuration, and Applied Configuration datastores can be mapped to the Running Configuration Datastore.

- If new YANG modules are supported that allow configuration and state nodes to be combined via the creation of system controlled entries, then <get> requests on the Running Configuration Datastore would also include any system controlled configuration entries along with decendent children nodes.

- The Operational State Datastore could be supported (for <get> and <get-config> requests only) and mapped to the Running Configuration Datatstore + config false nodes.

7.3. Implications for RESTCONF

The proposed revised datastore definitions have the following implications on RESTCONF:

- Support for explicit datastores is added to RESTCONF:
  - A new URL would be provided to allow for datastore specific access (e.g. "/restconf/ds/<datastore>/"
  - GET requests are supported on the new datastores, the content parameter could also be supported, but is pretty meaningless.
  - PUT, POST and PATCH are supported on the Persistent Config Datastore and Ephemeral Config Datastores, which are writable by clients.
  - It is an implementation choice whether devices support Intended Config and Applied Config as named, client accessable, datastores.
  - A new RESTCONF capability would indicate which new configuration datastores are supported by the device.
- Support for the new Operational State Datastore is added to RESTCONF:
  - GET requests return all the data from the datastore, and the content parameter used to choose whether config true, config false, or all nodes are returned.
  - PUT, POST and PATCH requests are not supported.
  - A new RESTCONF capability would indicate that the device supports the Operational State Datastore.
- Requests on the default RESTCONF datastore URL ("/restconf/data"):

  * PUT, POST and PATCH requests are handled as if they were made on the Persistent Configuration Datastore.

  * GET requests would be handled as if they were made on the Operational State Datastore.

7.3.1. Implications for Opstate Unaware Devices

An opstate unaware device does not have to make any changes, but a device could add support for the following to maximize interoperability:

- Support could be added for the named Persistent Configuration, Intended Configuration, and Applied Configuration datastores which would be handled as if the request was made directly on "/restconf/data".

- If new YANG modules are supported that allow configuration and state nodes to be combined via the creation of system controlled entries, then GET requests on "/restconf/data" would also include any system controlled configuration entries along with decendant children nodes.

- The Operational State Datastore could be supported for GET requests, and mapped to "/restconf/data".

8. Changes from previous version

Changes from previous versions:

- Changes from version -00:

  * The entire draft has been quite heavily restructured with an effort to make it easier to read

  * A definition of "abstract datastores" is given, with usage restricted to intended configuration, applied configuration, and running configuration

  * "System Controlled Configuration" and "System Controlled State" are no longer modelled as separate abstract datastores

  * The main diagram has been updated to make the relationship between the datastores clearer and more simple

  * Added support for schema labelling of "statistics" data nodes as part of the Operational State Datastore

* Added support for schema labelling of "ephemeral state" data nodes as part of the Operational State Datastore

9. Acknowledgements

This document is not solely the authors own work, but instead represents one view from the discussions to find a consensual solution to the operational state problem. It takes ideas from many people and uses parts of solutions described in the various internet drafts listed below. In particular, this document is an alternative to the revised datastore model described in draft-schoenw-netmod-revised-datastores [I-D.schoenw-netmod-revised-datastores], and reuses some of the structure and text from that document.

Work from the following internet drafts have helped form the basis of the datastore solution described in this draft:

- draft-bjorklund-netmod-operational [I-D.bjorklund-netmod-operational]
- draft-ietf-netmod-opstate-reqs [I-D.ietf-netmod-opstate-reqs]
- draft-kwatsen-netmod-opstate [I-D.kwatsen-netmod-opstate]
- draft-openconfig-netmod-opstate [I-D.openconfig-netmod-opstate]
- draft-schoenw-netmod-revised-datastores [I-D.schoenw-netmod-revised-datastores]
- draft-wilton-netmod-opstate-yang [I-D.wilton-netmod-opstate-yang]
- draft-ietf-i2rs-ephemeral-state [I-D.ietf-i2rs-ephemeral-state]

The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- Lou Berger, Labn Consulting, <lberger@labn.net>
- Andy Biermann, YumaWorks, <andy@yumaworks.com>
- Martin Bjorklund, Tail-f Systems, <mbj@tail-f.com>
- Susan Hares, Huawei, <shares@ndzh.com>
- Jeff Haas, Juniper Networks, <jhaas@juniper.net>
- Marcus Hines, Google, <hines@google.com>

- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Juergen Schoenwaelder, Jacobs University Bremen <j.schoenwaelder@jacobs-university.de>
- Anees Shaikh, Google, <aashaikh@google.com>
- Rob Shakir, Jive Communications, <rjs@rob.sh>
- Kent Watsen, Juniper Networks, <kwatsen@juniper.net>

The author would also like the thank the following people who have kindly provided feedback on this draft: Matt Hall, Einar Nilsen-Nygaard.

10. IANA Considerations

None

11. Security Considerations

This document discusses a conceptual model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

12. References

12.1. Normative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-14 (work in progress), June 2016.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, DOI 10.17487/RFC3535, May 2003, <http://www.rfc-editor.org/info/rfc3535>.

[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>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

12.2. Informative References

[I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-10 (work in progress), June 2016.

[I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[I-D.schoenw-netmod-revised-datastores]
Schoenwaelder, J., "A Revised Conceptual Model for YANG Datastores", draft-schoenw-netmod-revised-datastores-01 (work in progress), June 2016.

[I-D.wilton-netconf-opstate-metadata]
Wilton, R., "Refined YANG datastores with Meta-data", draft-wilton-netconf-opstate-metadata-00 (work in progress), July 2016.

[I-D.wilton-netmod-opstate-yang]
Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.

Author's Address

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com