NETCONF Data Modeling Language
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2017

L. Bertz
Sprint

March 9, 2017

YANG extension for Common Augmentations
draft-bertz-netmod-commonaugment-01

Abstract

This document defines a YANG extension to convey common schema augmentations.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November

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

1. Introduction

This document provides a mechanism for the specification of an augmentation at multiple locations of a YANG schema using the extension statement as defined in Section 7.15 of [RFC7950].

YANG extensions commonly add one augmentation to a common structure. Figure 1 shows a small module and another that will be used to extend it.

```
... basemodule {
  prefix b;
  ...
  grouping base_element {
      ...
  }

  rpc my_rpc {
    input {
      uses b:base_element;
    }
    output {
      uses b:base_element;
    }
  }

  rpc my_bestrpc {
    input {
      uses b:base_element;
    }
  }
}

... newstuffmodule {
  grouping new_things {
    ...
  }
}
```

Figure 1: Reference Modules

When extending my_rpc in the base module by adding new_things to the base element, one must use two augment statements. If the author of the newstuff module wishes to extend base_element then they must write an augment statement for every schema location it which it appears.

An extension with the key word "also-augments" is proposed that allows one augment statement to be used for augmentations.

Figure 2 shows the traditional and common augment examples.

```
module old_way {
  import base { prefix base_module; }
  import newstuff { prefix new_module; }
  augment "/base_module:my_rpc/base_element/input" {
    uses new_module:new_things;
  }
  augment "/base_module:my_rpc/base_element/output" {
    uses new_module:new_things;
  }
  augment "/base_module:my_bestrpc/base_element/input" {
    uses new_module:new_things;
  }
}

module new_way {
   import base { prefix base_module; }
   import newstuff { prefix new_module; }
   import commonaugment { prefix c; }

   augment "/base_module:my_rpc/base_element/input" {
     uses new_module:new_things;

     c:also-augments "/base_module:my_rpc/base_element/output";
     c:also-augments "/base_module:my_bestrpc/base_element/input";
   }
 }
```

Figure 2: Augment Options

The also-augments statement takes a single argument that is the same one defined for the augment statement [RFC7950].

This has two purposes:

Reduces the amount of YANG written when common augmentations are invovled.

Provides a hint to the tools that translate YANG to code that a common augmentation exists.

The importance of the second benefit for is significant as it permits them to leverage any common structure or inheritence functions.

2. 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].

3. Overview

This module defines only the also-augmennts extension. This extension is ONLY meaningful as a child of an augment statement and is otherwise ignored. It is semantically equivalent to writing an augment statement whose target node is the argument of the also-augment node. This is shown in Figure 3.

```
augment "/base_module:my_rpc/base_element/input" {
  uses new_module:new_things;
}
augment "/base_module:my_rpc/base_element/output" {
  uses new_module:new_things;
}

... semantically equivalent version below ...

augment "/base_module:my_rpc/base_element/input" {
  uses new_module:new_things;

  c:also-augments "/base_module:my_rpc/base_element/output";
}
```

Figure 3: Statement Equivalence of also-augments

Multiple also-augments statements MAY appear in a single augment statement.

This statement MAY appear in a module that is intended to be backwards compatible as shown in Figure 4. In such cases, it is used as a hint to readers and compilation software that the augmentations are intended to be the same augmentation at multiple schema tree locations.

```
        augment "/base_module:my_rpc/base_element/input" {
          uses new_module:new_things;
          c:also-augments "/base_module:my_rpc/base_element/output";
        }
        augment "/base_module:my_rpc/base_element/output" {
          uses new_module:new_things;
        }
```

Figure 4: Backwards compatible usage of also-augment

4. Module Definition

This section contains the module definition.

```
<CODE BEGINS> file "ietf-commonaugment@2017-03-04.yang"
module ietf-commonaugment {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-commonaugment";
    prefix commonaug;

    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:   Lyle Bertz
                  <mailto:lyleb551144@gmail.com>";

    description
    "This module contains YANG definition for
     common augments.

     Copyright (c) 2016 IETF Trust and the persons identified as the
     document authors. All rights reserved.

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

```
        respect to this document. Code Components extracted from this
        document must include Simplified BSD License text as described
        in Section 4.e of the Trust Legal Provisions and are provided
        without warranty as described in the Simplified BSD License.";

      revision 2017-03-04 {
          description "Initial Revision.";
          reference "draft-bertz-netmod-commonaugment-00";
      }

      extension also-augments {
        argument "target-node";
        description
          "The argument is a string that identifies a node in the
          schema tree and is the same argument that is defined for
          the YANG argument statement";
      }
  }
  <CODE ENDS>
```

5. IANA Considerations

This document registers six URIs in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations have been made.

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

This document registers the following YANG module in the "YANG Module Names" registry [RFC7950].

```
        name:         ietf-dmm-fpc
        namespace:    urn:ietf:params:xml:ns:yang:commonaug
        prefix:       commonaug
        reference:    TBD1
```

6. Security Considerations

This document provides an alternative mechanism to express augmentations that can exist in a YANG module. It does not provide new informaiton elements to the module but does provide a hint as to the commonality of a set of augment statements. This, however, SHOULD have already been described in the module’s definition or specification. Thus, the use of this statement does not yield new information.

7. References

7.1. 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>.

[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>.

7.2. 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>.

Author’s Address

Lyle Bertz
Sprint
6220 Sprint Parkway
Overland Park, KS 66251
United States

Email: lylebe551144@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

M. Bjorklund
Tail-f Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
March 13, 2017

YANG Tree Diagrams
draft-bjorklund-netmod-yang-tree-diagrams-00

Abstract

This document captures the current syntax used in YANG module Tree Diagrams. The purpose of the document is to provide a single location for this definition. This syntax may be updated from time to time based on the evolution of the YANG language.

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, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
2. Tree Diagram Syntax 2
3. Next Steps 3
4. IANA Considerations 4
5. Informative References 4
Authors' Addresses 4

1. Introduction

YANG Tree Diagrams were first published in [RFC7223]. Such diagrams are commonly used to provided a simplified graphical representation of a data model and can be automatically generated via tools such as "pyang". (See <https://github.com/mbj4668/pyang>). This document provides the syntax used in YANG Tree Diagrams. It is expected that this document will be updated or replaced as changes to the YANG language, see [RFC7950], necessitate.

Today's Common practice is include the definition of the syntax used to represent a YANG module in every document that provides a tree diagram. This practice has several disadvantages and the purpose of the document is to provide a single location for this definition. It is not the intent of this document to restrict future changes, but rather to ensure such changes are easily identified and suitably agreed upon.

An example tree diagram can be found in [RFC7223] Section 3. A portion of which follows:

```
+--rw interfaces
|  +--rw interface* [name]
|     +--rw name                        string
|     +--rw description?                string
|     +--rw type                        identityref
|     +--rw enabled?                    boolean
|     +--rw link-up-down-trap-enable?   enumeration
```

The remainder of this document contains YANG Tree Diagram syntax based on output from pyang version 1.7.1.

2. Tree Diagram Syntax

This section provides the meaning of the symbols used in YANG Tree diagrams.

Each node in a YANG module 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 and actions
    -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, choice, anydata or anyxml
    !  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 "{...}?"
```

3. Next Steps

Authors' note: This draft is currently a bit rough. The next/future version is expected to add text covering different types of trees and when to use them; for example, complete module trees, subtrees, trees for groupings etc. Maybe also how to deal with the limited line lengths in RFCs.

4. IANA Considerations

There are no IANA requests or assignments included in this document.

5. Informative References

[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>.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: March 23, 2017

A. Clemm
J. Medved
E. Voit
Cisco Systems

September 19, 2016

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-05.txt

Abstract

This document introduces capabilities that allow YANG datastores to reference and incorporate information from remote datastores. This is accomplished by extending YANG with the ability to define mount points that reference data nodes in another YANG subtree, by subsequently allowing those data nodes to be accessed by client applications as if part of an alternative data hierarchy, and by providing the necessary means to manage and administer those mount points. Two flavors are defined: Alias-Mount allows to mount local subtrees, while Peer-Mount allows subtrees to reside on and be authoritatively owned by a remote server. YANG-Mount facilitates the development of applications that need to access data that transcends individual network devices while improving network-wide object consistency, or that require an aliasing capability to be able to create overlay structures for YANG data.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 23, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction 3
1.1. Overview 3
1.2. Examples 5
2. Definitions and Acronyms 7
3. Example scenarios 8
3.1. Network controller view 8
3.2. Consistent network configuration 10
4. Operating on mounted data 11
4.1. General principles 12
4.2. Data retrieval 12
4.3. Other operations 13
4.4. Other considerations 13
5. Data model structure 14
5.1. YANG mountpoint extensions 14
5.2. YANG structure diagrams 15
5.3. Mountpoint management 15
5.4. Caching 17
5.5. Other considerations 18
5.5.1. Authorization 18

5.5.2. Datastore qualification 18
5.5.3. Mount cascades 18
5.5.4. Implementation considerations 19
5.5.5. Modeling best practices 20
6. Datastore mountpoint YANG module 20
7. Security Considerations 28
8. Acknowledgements 28
9. References 29
9.1. Normative References 29
9.2. Informative References 30
Appendix A. Example 31
Authors' Addresses 35

1. Introduction

1.1. Overview

This document introduces a new capability that allows YANG datastores [RFC6020] to incorporate and reference information from other YANG subtrees. The capability allows a client application to retrieve and have visibility of that YANG data as part of an alternative structure. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node in a primary datastore to serve as a "mount point" under which a subtree with YANG data can be mounted. This way, data nodes from another subtree can be inserted into an alternative data hierarchy, arranged below local data nodes. To the user, this provides visibility to data from other subtrees, rendered in a way that makes it appear largely as if it were an integral part of the datastore. This enables users to retrieve local "native" as well as mounted data in integrated fashion, using e.g. Netconf [RFC6241] or Restconf [I-D.ietf-netconf-restconf] data retrieval primitives. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user's machine.

Two variants of YANG-Mount are introduced, which build on one another:

- Alias-Mount allows mountpoints to reference a local YANG subtree residing on the same server. It provides effectively an aliasing capability, allowing for an alternative hierarchy and path for the same YANG data.

- Peer-Mount allows mountpoints to reference a remote YANG subtree, residing on a different server. It can be thought of as an extension to Alias-Mount, in which a remote server can be

specified. Peer-Mount allows a server to effectively provide a federated datastore, including YANG data from across the network.

In each case, mounted data is authoritatively owned by the server that it is a part of. Validation of integrity constraints apply to the authoritative copy; mounting merely provides a different view of the same data. It does not impose additional constraints on that same data; however, mounted data may be referred to from other data nodes. The mountpoint concept applies in principle to operations beyond data retrieval, i.e. to configuration, RPCs, and notifications. However, support for such operations involves additional considerations, for example if support for configuration transactions and locking (which might now apply across the network) were to be provided. While it is conceivable that additional capabilities for operations on mounted information are introduced at some point in time, their specification is beyond the scope of this specification.

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

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

The capability of allowing the mounting of information from other subtrees is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module and its server (i.e. the "receivers" or "consumers" of the mounted information) need to be aware of the concepts introduced here. Mounting is transparent to the "providers" of the mounted information and models that are being

mounted; any data nodes or subtrees within any YANG model can be mounted.

Alias-Mount and Peer-Mount build on top of each other. It is possible for a server to support Alias-Mount but not Peer-Mount. In essence, Peer-Mount requires an additional parameter that is used to refer to the target system. This parameter does not need to be supported if only Alias-Mount is provided.

Finally, it should be mentioned that Alias-Mount and Peer-Mount are not to be confused with the ability to mount a schema, aka Schema Mount. A Schema Mount allows to instantiate an existing model definition underneath a mount point, not reference a set of YANG data that has already been instantiated somewhere else. In that sense, Schema-Mount resembles more a "grouping" concept that allows to reuse an existing definition in a new context, as opposed to referencing and incorporating existing instance information into a new context.

1.2. Examples

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

One category of applications that can leverage this capability are network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface data, both operational data (e.g. various types of interface statistics) and configuration data, such as defined in [RFC7223]. This data will be contained in a top-level container ("interfaces", in this particular case) in a network element datastore. The controller may need to provide its clients a view on interface data from multiple devices under its scope of control. One way of to do so would involve organizing the data in a list with separate list elements for each device. However, this in turn would require introduction of redundant YANG modules that

effectively replicate the same interface data save for differences in hierarchy.

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

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

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

It should be noted that for these and many other applications merely having a view of the remote information is sufficient. It allows to define consolidated views of information without the need for replicating data and models that have already been defined, to audit information, and to validate consistency of configurations across a

network. Only retrieval operations are required; no operations that involve configuring remote data are involved.

2. Definitions and Acronyms

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

DHCP: Dynamic Host Configuration Protocol.

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

Datastore-push: A mechanism that allows a client to subscribe to updates from a datastore, which are then automatically pushed by the server to the client.

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

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

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

Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

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

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

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

3.1. Network controller view

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

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

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element’s, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data inconsistency and staleness. Equally important, it would lead to the need to define redundant data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

For example, "system"-level information of a network element would most naturally placed into a top-level container at that network element’s datastore. At the same time, the same information in the context of the overall network, such as maintained by a controller, might better be provided in a list. For example, the controller might maintain a list with a list element for each network element, underneath which the network element’s system-level information is

contained. However, the containment structure of data nodes in a module, once defined, cannot be changed. This means that in the context of a network controller, a second module that repeats the same system-level information would need to be defined, implemented, and maintained. Any augmentations that add additional system-level information to the original module will likewise need to be redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions just to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore (the "primary" datastore) could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore. The data from the mounted subtrees is then accessible to clients of the primary datastore using the usual data retrieval operations.

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

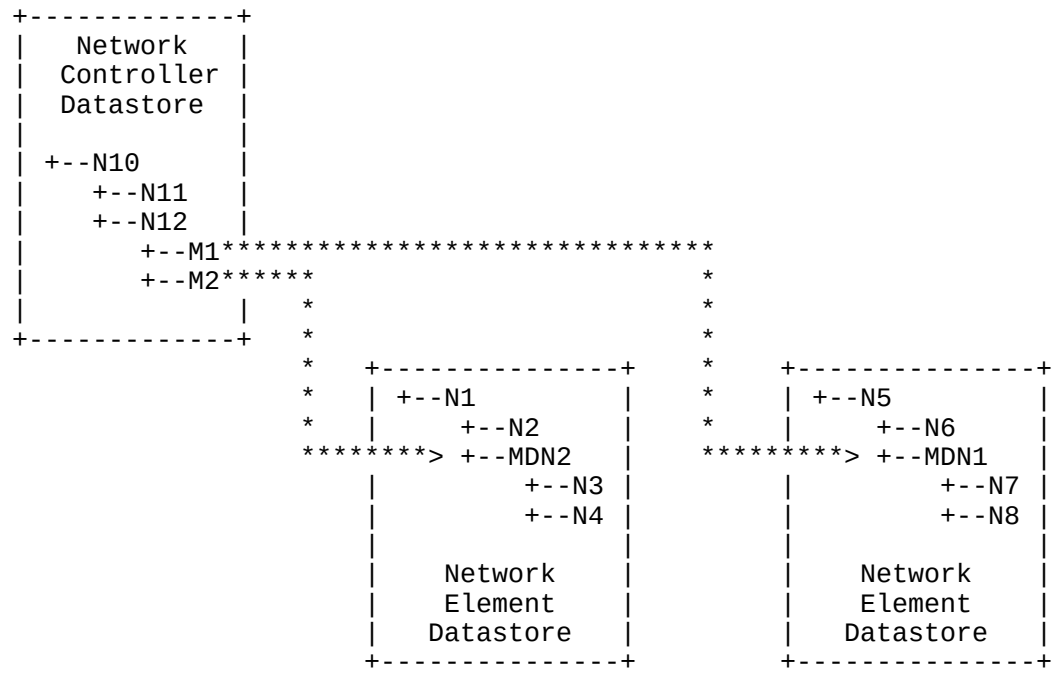


Figure 1: Network controller mount topology

3.2. Consistent network configuration

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

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

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

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

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings. (Note that there is no reason why the Network Controller Datastore in this figure could not simply reside on a network element itself; the division of responsibilities is a logical one.

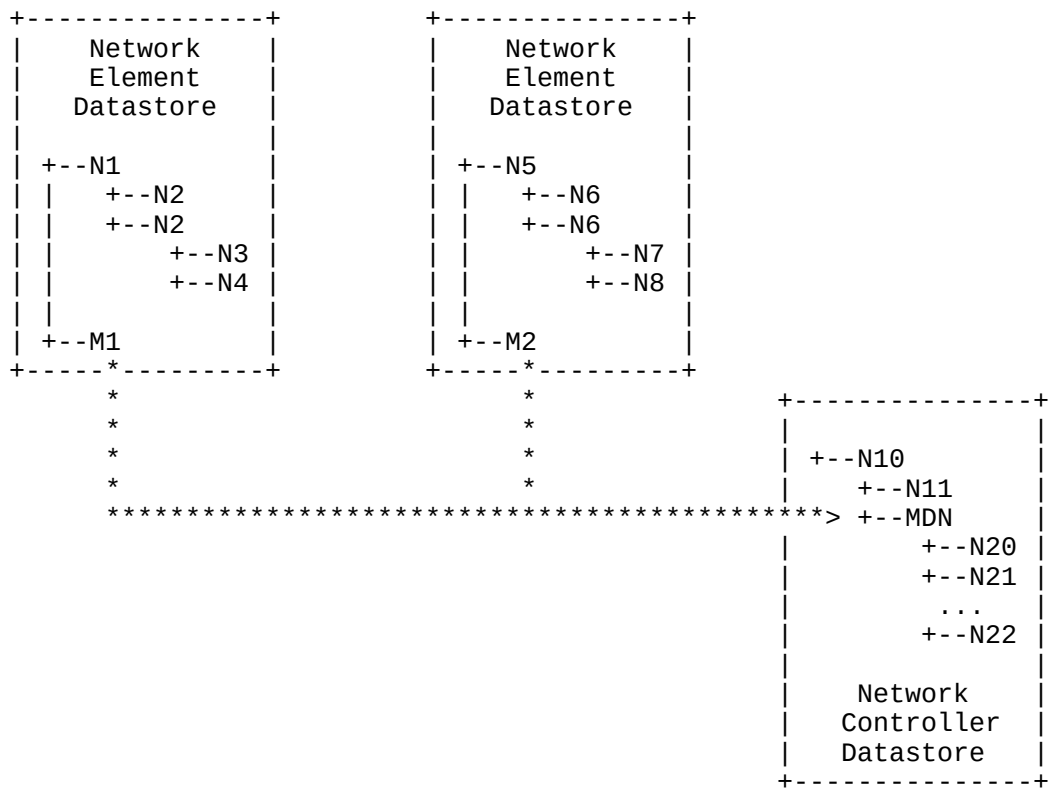


Figure 2: Distributed config settings topology

4. Operating on mounted data

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

4.1. General principles

The first thing that should be noted about these operations flows concerns the fact that a mount client essentially constitutes a special management application that interacts with a subtree to render the data of that subtree as an alternative tree hierarchy. In the case of Alias-Mount, both original and alternative tree are maintained by the same server, which in effect provides alternative paths to the same data. In the case of Peer-Mount, the mount client constitutes in effect another application, with the remote system remaining the authoritative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate the specific needs of the mount client to make it more efficient, the fact that another system decides to expose a certain "view" of that data is fundamentally not the remote system’s concern.

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

4.2. Data retrieval

Data retrieval operations are the only category of operations that is supported for peer-mounted information. In that case, a Netconf "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system’s subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

4.3. Other operations

The fact that only data retrieval operations are the only category of operations that are supported for peer-mounted information does not preclude other operations to be applied to datastore subtrees that contain mountpoints and peer-mounted information. Peer-mounted information is simply transparent to those operations. When an operation is applied to a subtree which includes mountpoints, mounted information is ignored for purposes of the operation. For example, for a Netconf "edit-config" operation that includes a subtree with a mountpoint, a server will ignore the data under the mountpoint and apply the operation only to the local configuration. Mounted data is "read-only" data. The server does not even need to return an error message that the operation could not be applied to mounted data; the mountpoint is simply ignored.

In principle, it is conceivable that operations other than data-retrieval are applied to mounted data as well. For example, an operation to edit configuration information might expect edits to be applied to remote systems as part of the operation, where the edited subtree involves mounted information. However, editing of information and "writing through" to remote systems potentially involves significant complexity, particularly if transactions and locking across multiple configuration items are involved. Support for such operations will require additional capabilities, specification of which is beyond the scope of this specification.

Likewise, YANG-Mount does not extend towards RPCs that are defined as part of YANG modules whose contents is being mounted. Support for RPCs that involve mounted portions of the datastore, while conceivable, would require introduction of an additional capability, whose definition is outside the scope of this specification.

By the same token, YANG-Mount does not extend towards notifications. It is conceivable to offer such support in the future using a separate capability, definition of which is once again outside the scope of this specification.

4.4. Other considerations

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

It is expected that some implementations will introduce caching schemes. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Caching is not required for YANG-mount to work - in which case access to mounted information is "on-demand", in which the authoritative data node always gets accessed. Whether to perform caching is a local implementation decision.

When caching is introduced, it can benefit from the ability to subscribe to updates on remote data by remote servers. Requirements for such a capability have been defined in [RFC7923]. Some optimizations to facilitate caching support will be discussed in section Section 5.4.

5. Data model structure

5.1. YANG mountpoint extensions

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

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

- The second extension, "subtree", serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression. This extension is used to define mountpoints for Alias-Mount, as well as Peer-Mount.

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

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

Only a single data node can be mounted at one time. While the mount target could refer to any data node, it is recommended that as a best

practice, the mount target SHOULD refer to a container. It is possible to maintain e.g. a list of mount points, with each mount point each of which has a mount target an element of a remote list. However, to avoid unnecessary proliferation of the number of mount points and associated management overhead, when data from lists or leaf-lists is to be mounted, a container containing the list respectively leaf-list SHOULD be mounted instead of individual list elements.

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

5.2. YANG structure diagrams

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

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

5.3. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

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

such as the target being unreachable or referring to an invalid subtree).

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

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

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

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

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

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

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

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

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

5.4. Caching

Under certain circumstances, it can be useful to maintain a cache of remote information. Instead of accessing the remote system, requests are served from a copy that is locally maintained. This is particularly advantageous in cases where data is slow changing, i.e. when there are many more "read" operations than changes to the underlying data node, and in cases when a significant delay were incurred when accessing the remote system, which might be prohibitive for certain applications. Examples of such applications are applications that involve real-time control loops requiring response times that are measured in milliseconds. However, as data nodes that are mounted from an authoritative datastore represent the "golden

copy", it is important that any modifications are reflected as soon as they are made.

It is a local implementation decision of mount clients whether to cache information once it has been fetched. However, in order to support more powerful caching schemes, it becomes necessary for the mount server to "push" information proactively. For this purpose, it is useful for the mount client to subscribe for updates to the mounted information at the mount server. A corresponding mechanism that can be leveraged for this purpose is specified in [I-D.ietf-netconf-yang-push].

Note that caching large mountpoints can be expensive. Therefore limiting the amount of data unnecessarily passed when mounting near the top of a YANG subtree is important. For these reasons, an ability to specify a particular caching strategy in conjunction with mountpoints can be desirable, including the ability to exclude certain nodes and subtrees from caching. According capabilities may be introduced in a future version of this draft.

5.5. Other considerations

5.5.1. Authorization

Access to mounted information is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

5.5.2. Datastore qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

5.5.3. Mount cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount

target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

5.5.4. Implementation considerations

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

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

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

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

- The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client's behalf. The communication between mount client and remote mount server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

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

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

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested

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

5.5.5. Modeling best practices

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

For those reasons, as a general best practice, models that make use of mount points SHOULD be defined in a way that minimizes the number of mountpoints required. Finely granular mounts, in which multiple mountpoints are maintained with the same remote system, each containing only very small data subtrees, SHOULD be avoided. For example, lists SHOULD only contain mountpoints when individual list elements are associated with different remote systems. To mount data from lists in remote datastores, a container node that contains all list elements SHOULD be mounted instead of mounting each list element individually. Likewise, instead of having mount points refer to nodes contained underneath choices, a mountpoint should refer to a container of the choice.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "ietf-mount@2016-09-19.yang"
module ietf-mount {
  namespace "urn:ietf:params:xml:ns:yang:ietf-mount";
  prefix mnt;

  import ietf-inet-types {
    prefix inet;
  }

  organization
```

```
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    contact
      "WG Web:   <http://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>

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

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

       Editor: Alexander Clemm
       <mailto:ludwig@clemm.org>

       Editor: Jan Medved
       <mailto:jmedved@cisco.com>

       Editor: Eric Voit
       <mailto:evoit@cisco.com>";
    description
      "This module provides a set of YANG extensions and definitions
       that can be used to mount information from remote datastores.";

    revision 2016-09-19 {
      description
        "Initial revision.";
      reference
        "draft-clemm-netmod-mount-05.txt";
    }

    extension mountpoint {
      argument name;
      description
        "This YANG extension is used to mount data from another
         subtree in place of the node under which this YANG extension
         statement is used.

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

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

         The extension can take two parameters, target and subtree,
         each defined as their own YANG extensions.
```

```
          For Alias-Mount, a mountpoint statement MUST contain a
          subtree statement for the mountpoint definition to be valid.
          For Peer-Mount, a mountpoint statement MUST contain both a
          target and a subtree substatement for the mountpoint
          definition to be valid.

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

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

          It is possible for the mounted subtree to in turn contain a
          mountpoint.  However, circular mount relationships MUST NOT
          be introduced. For this reason, a mounted subtree MUST NOT
          contain a mountpoint that refers back to the mounting system
          with a mount target that directly or indirectly contains the
          originating mountpoint.";
      }

      extension target {
        argument target-name;
        description
          "This YANG extension is used to perform a Peer-Mount.
           It is used to specify a remote target system from which to
           mount a datastore subtree.  This YANG
           extension takes one argument which specifies the remote
           system. In general, this argument will contain the name of
           a data node that contains the remote system information. It
           is recommended that the reference data node uses the
           mount-target grouping that is defined further below in this
           module.

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

      extension subtree {
        argument subtree-path;
        description
          "This YANG extension is used to specify a subtree in a
           datastore that is to be mounted.  This YANG extension takes
           one argument which specifies the path to the root of the
```

```
          subtree. The root of the subtree SHOULD represent an
          instance of a YANG container.  However, it MAY represent
          also another data node.

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

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

      typedef mount-status {
        type enumeration {
          enum "ok" {
            description
              "Mounted";
          }
          enum "no-target" {
            description
              "The argument of the mountpoint does not define a
               target system";
          }
          enum "no-subtree" {
            description
              "The argument of the mountpoint does not define a
                root of a subtree";
          }
          enum "target-unreachable" {
            description
              "The specified target system is currently
               unreachable";
          }
          enum "mount-failure" {
            description
              "Any other mount failure";
          }
          enum "unmounted" {
            description
              "The specified mountpoint has been unmounted as the
               result of a management operation";
          }
        }
        description
          "This type is used to represent the status of a
```

```
          mountpoint.";
      }

      typedef subtree-ref {
        type string;
        description
          "This string specifies a path to a datanode. It corresponds
           to the path substatement of a leafref type statement.  Its
           syntax needs to conform to the corresponding subset of the
           XPath abbreviated syntax. Contrary to a leafref type,
           subtree-ref allows to refer to a node in a remote datastore.
           Also, a subtree-ref refers only to a single node, not a list
           of nodes.";
      }

      grouping mount-monitor {
        description
          "This grouping contains data nodes that indicate the
           current status of a mount point.";
        leaf mount-status {
          type mount-status;
          config false;
          description
            "Indicates whether a mountpoint has been successfully
             mounted or whether some kind of fault condition is
             present.";
        }
      }

      grouping mount-target {
        description
          "This grouping contains data nodes that can be used to
           identify a remote system from which to mount a datastore
           subtree.";
        container mount-target {
          description
            "A container is used to keep mount target information
             together.";
          choice target-address-type {
            mandatory true;
            description
              "Allows to identify mount target in different ways,
               i.e. using different types of addresses.";
            case IP {
              leaf target-ip {
                type inet:ip-address;
                description
                  "IP address identifying the mount target.";
```

```
            }
          }
          case URI {
            leaf uri {
              type inet:uri;
              description
                "URI identifying the mount target";
            }
          }
          case host-name {
            leaf hostname {
              type inet:host;
              description
                "Host name of mount target.";
            }
          }
          case node-ID {
            leaf node-info-ref {
              type subtree-ref;
              description
                "Node identified by named subtree.";
            }
          }
          case other {
            leaf opaque-target-ID {
              type string;
              description
                "Catch-all; could be used also for mounting
                 of data nodes that are local.";
            }
          }
        }
      }
    }

    grouping mount-policies {
      description
        "This grouping contains data nodes that allow to configure
         policies associated with mountpoints.";
      leaf manual-mount {
        type empty;
        description
          "When present, a specified mountpoint is not
           automatically mounted when the mount data node is
           created, but needs to mounted via specific RPC
           invocation.";
      }
      leaf retry-timer {
```

```
        type uint16;
        units "seconds";
        description
          "When specified, provides the period after which
           mounting will be automatically reattempted in case of a
           mount status of an unreachable target";
      }
      leaf number-of-retries {
        type uint8;
        description
          "When specified, provides a limit for the number of
           times for which retries will be automatically
           attempted";
      }
    }

    rpc mount {
      description
        "This RPC allows an application or administrative user to
         perform a mount operation.  If successful, it will result in
         the creation of a new mountpoint.";
      input {
        leaf mountpoint-id {
          type string {
            length "1..32";
          }
          description
            "Identifier for the mountpoint to be created.
             The mountpoint-id needs to be unique;
             if the mountpoint-id of an existing mountpoint is
             chosen, an error is returned.";
        }
      }
      output {
        leaf mount-status {
          type mount-status;
          description
            "Indicates if the mount operation was successful.";
        }
      }
    }
    rpc unmount {
      description
        "This RPC allows an application or administrative user to
         unmount information from a remote datastore.  If successful,
         the corresponding mountpoint will be removed from the
         datastore.";
      input {
```

```
        leaf mountpoint-id {
          type string {
            length "1..32";
          }
          description
            "Identifies the mountpoint to be unmounted.";
        }
      }
      output {
        leaf mount-status {
          type mount-status;
          description
            "Indicates if the unmount operation was successful.";
        }
      }
    }
    container mount-server-mgmt {
      if-feature mount-server-mgmt;
      description
        "Contains information associated with managing the
         mountpoints of a datastore.";
      container mountpoints {
        description
          "Keep the mountpoint information consolidated
           in one place.";
        list mountpoint {
          key "mountpoint-id";
          description
            "There can be multiple mountpoints.
             Each mountpoint is represented by its own
             list element.";
          leaf mountpoint-id {
            type string {
              length "1..32";
            }
            description
              "An identifier of the mountpoint.
               RPC operations refer to the mountpoint
               using this identifier.";
          }
          leaf mountpoint-origin {
            type enumeration {
              enum "client" {
                description
                  "Mountpoint has been supplied and is
                   manually administered by a client";
              }
              enum "auto" {
```

```
                description
                  "Mountpoint is automatically
                   administered by the server";
              }
            }
            config false;
            description
              "This describes how the mountpoint came
               into being.";
          }
          leaf subtree-ref {
            type subtree-ref;
            mandatory true;
            description
              "Identifies the root of the subtree in the
               target system that is to be mounted.";
          }
          uses mount-target;
          uses mount-monitor;
          uses mount-policies;
        }
      }
      container global-mount-policies {
        description
          "Provides mount policies applicable for all mountpoints,
           unless overridden for a specific mountpoint.";
        uses mount-policies;
      }
    }
  }
```

<CODE ENDS>

7. Security Considerations

TBD

8. Acknowledgements

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

9. References

9.1. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <http://www.rfc-editor.org/info/rfc2866>.

[RFC3768] Hinden, R., Ed., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, DOI 10.17487/RFC3768, April 2004, <http://www.rfc-editor.org/info/rfc3768>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[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>.

[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>.

[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>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <http://www.rfc-editor.org/info/rfc7923>.

9.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restonf-16 (work in progress), August 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Gonzalez Prieto, A., Voit, E., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-03 (work in progress), June 2016.

[I-D.voit-netmod-yang-mount-requirements]
Voit, E., Clemm, A., and S. Mertens, "Requirements for mounting of local and remote YANG subtrees", draft-voit-netmod-yang-mount-requirements-00 (work in progress), March 2016.

Appendix A. Example

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

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

The controller network model consists of the following key components:

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

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

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

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

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

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

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

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

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

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

Interface information from NE1 datastore:

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

Interface information from NE2 datastore:

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

NC datastore with mounted interface information from NE1 and NE2:

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

Authors' Addresses

Alexander Clemm
Cisco Systems

EMail: ludwig@clemm.org

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: October 1, 2017

A. Clemm
Huawei
E. Voit
J. Medved
Cisco Systems

March 30, 2017

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-06.txt

Abstract

This document introduces capabilities that allow YANG datastores to reference and incorporate information from remote datastores. This is accomplished by extending YANG with the ability to define mount points that reference data nodes in another YANG subtree, by subsequently allowing those data nodes to be accessed by client applications as if part of an alternative data hierarchy, and by providing the necessary means to manage and administer those mount points. Two flavors are defined: Alias-Mount allows to mount local subtrees, while Peer-Mount allows subtrees to reside on and be authoritatively owned by a remote server. YANG-Mount facilitates the development of applications that need to access data that transcends individual network devices while improving network-wide object consistency, or that require an aliasing capability to be able to create overlay structures for YANG data.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 1, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction 3
1.1. Overview 3
1.2. Examples 5
2. Definitions and Acronyms 7
3. Example scenarios 7
3.1. Network controller view 8
3.2. Consistent network configuration 10
4. Operating on mounted data 11
4.1. General principles 11
4.2. Data retrieval 12
4.3. Other operations 12
4.4. Other considerations 13
5. Data model structure 14
5.1. YANG mountpoint extensions 14
5.2. YANG structure diagrams 15
5.3. Mountpoint management 15
5.4. Caching 17
5.5. Other considerations 18
5.5.1. Authorization 18

5.5.2. Datastore qualification 18
5.5.3. Mount cascades 18
5.5.4. Implementation considerations 19
5.5.5. Modeling best practices 20
6. Datastore mountpoint YANG module 20
7. Security Considerations 28
8. Acknowledgements 28
9. Normative References 28
Appendix A. Example 30
Authors' Addresses 34

1. Introduction

1.1. Overview

This document introduces a new capability that allows YANG datastores [RFC7950] to incorporate and reference information from other YANG subtrees. The capability allows a client application to retrieve and have visibility of that YANG data as part of an alternative structure. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node in a primary datastore to serve as a "mount point" under which a subtree with YANG data can be mounted. This way, data nodes from another subtree can be inserted into an alternative data hierarchy, arranged below local data nodes. To the user, this provides visibility to data from other subtrees, rendered in a way that makes it appear largely as if it were an integral part of the datastore. This enables users to retrieve local "native" as well as mounted data in integrated fashion, using e.g. Netconf [RFC6241] or Restconf [RFC8040] data retrieval primitives. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user's machine.

Two variants of YANG-Mount are introduced, which build on one another:

- Alias-Mount allows mountpoints to reference a local YANG subtree residing on the same server. It provides effectively an aliasing capability, allowing for an alternative hierarchy and path for the same YANG data.

- Peer-Mount allows mountpoints to reference a remote YANG subtree, residing on a different server. It can be thought of as an extension to Alias-Mount, in which a remote server can be specified. Peer-Mount allows a server to effectively provide a federated datastore, including YANG data from across the network.

In each case, mounted data is authoritatively owned by the server that it is a part of. Validation of integrity constraints apply to the authoritative copy; mounting merely provides a different view of the same data. It does not impose additional constraints on that same data; however, mounted data may be referred to from other data nodes. The mountpoint concept applies in principle to operations beyond data retrieval, i.e. to configuration, RPCs, and notifications. However, support for such operations involves additional considerations, for example if support for configuration transactions and locking (which might now apply across the network) were to be provided. While it is conceivable that additional capabilities for operations on mounted information are introduced at some point in time, their specification is beyond the scope of this specification.

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

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

The capability of allowing the mounting of information from other subtrees is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module and its server (i.e. the "receivers" or "consumers" of the mounted information) need to be aware of the concepts introduced here. Mounting is transparent to the "providers" of the mounted information and models that are being mounted; any data nodes or subtrees within any YANG model can be mounted.

Alias-Mount and Peer-Mount build on top of each other. It is possible for a server to support Alias-Mount but not Peer-Mount. In essence, Peer-Mount requires an additional parameter that is used to refer to the target system. This parameter does not need to be supported if only Alias-Mount is provided.

Finally, it should be mentioned that Alias-Mount and Peer-Mount are not to be confused with the ability to mount a schema, aka Schema Mount. A Schema Mount allows to instantiate an existing model definition underneath a mount point, not reference a set of YANG data that has already been instantiated somewhere else. In that sense, Schema-Mount resembles more a "grouping" concept that allows to reuse an existing definition in a new context, as opposed to referencing and incorporating existing instance information into a new context.

1.2. Examples

The ability to mount data from remote datastores is useful to address various problems that several categories of applications are faced with.

One category of applications that can leverage this capability are network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface data, both operational data (e.g. various types of interface statistics) and configuration data, such as defined in [RFC7223]. This data will be contained in a top-level container ("interfaces", in this particular case) in a network element datastore. The controller may need to provide its clients a view on interface data from multiple devices under its scope of control. One way of to do so would involve organizing the data in a list with separate list elements for each device. However, this in turn would require introduction of redundant YANG modules that effectively replicate the same interface data save for differences in hierarchy.

By directly mounting information from network element datastores, the controller does not need to replicate the same information from

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

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

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

It should be noted that for these and many other applications merely having a view of the remote information is sufficient. It allows to define consolidated views of information without the need for replicating data and models that have already been defined, to audit information, and to validate consistency of configurations across a network. Only retrieval operations are required; no operations that involve configuring remote data are involved.

2. Definitions and Acronyms

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

DHCP: Dynamic Host Configuration Protocol.

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

Datastore-push: A mechanism that allows a client to subscribe to updates from a datastore, which are then automatically pushed by the server to the client.

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

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

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

Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

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

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

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

3.1. Network controller view

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

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

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element's, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data inconsistency and staleness. Equally important, it would lead to the need to define redundant data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

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

redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions just to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore (the "primary" datastore) could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore. The data from the mounted subtrees is then accessible to clients of the primary datastore using the usual data retrieval operations.

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

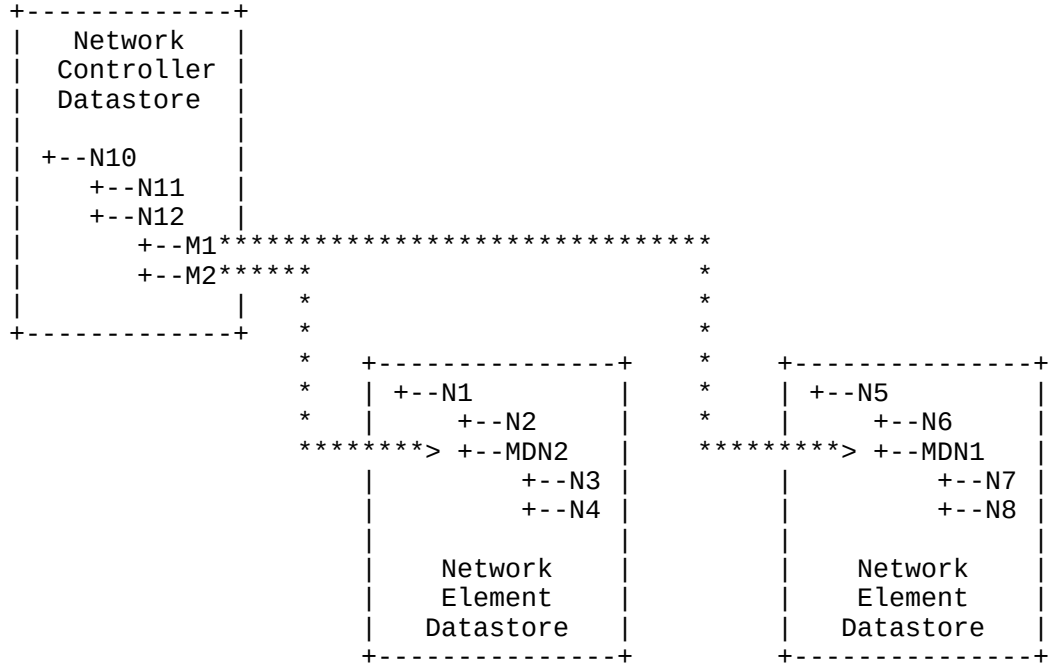


Figure 1: Network controller mount topology

3.2. Consistent network configuration

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

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

In many ways, this category of applications is an inverse of the previous category: Whereas in the network controller case data from many different datastores would be mounted into the same datastore with multiple mountpoints, in this case many elements, each with their own datastore, mount the same remote datastore, which is then mounted by many different systems.

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings. (Note that there is no reason why the Network Controller Datastore in this figure could not simply reside on a network element itself; the division of responsibilities is a logical one.

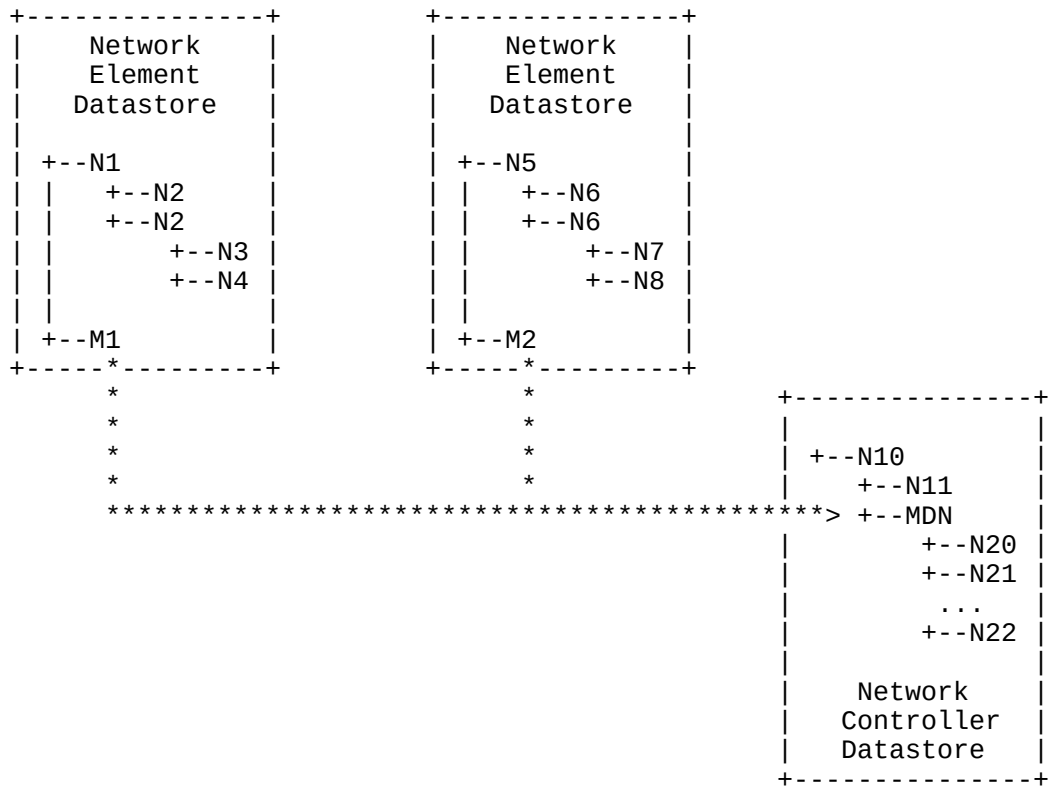


Figure 2: Distributed config settings topology

4. Operating on mounted data

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

4.1. General principles

The first thing that should be noted about these operations flows concerns the fact that a mount client essentially constitutes a special management application that interacts with a subtree to render the data of that subtree as an alternative tree hierarchy. In the case of Alias-Mount, both original and alternative tree are maintained by the same server, which in effect provides alternative paths to the same data. In the case of Peer-Mount, the mount client constitutes in effect another application, with the remote system remaining the authoritative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate

the specific needs of the mount client to make it more efficient, the fact that another system decides to expose a certain "view" of that data is fundamentally not the remote system's concern.

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

4.2. Data retrieval

Data retrieval operations are the only category of operations that is supported for peer-mounted information. In that case, a Netconf "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system's subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

4.3. Other operations

The fact that only data retrieval operations are the only category of operations that are supported for peer-mounted information does not preclude other operations to be applied to datastore subtrees that contain mountpoints and peer-mounted information. Peer-mounted information is simply transparent to those operations. When an operation is applied to a subtree which includes mountpoints, mounted information is ignored for purposes of the operation. For example, for a Netconf "edit-config" operation that includes a subtree with a mountpoint, a server will ignore the data under the mountpoint and apply the operation only to the local configuration. Mounted data is "read-only" data. The server does not even need to return an error message that the operation could not be applied to mounted data; the mountpoint is simply ignored.

In principle, it is conceivable that operations other than data-retrieval are applied to mounted data as well. For example, an operation to edit configuration information might expect edits to be applied to remote systems as part of the operation, where the edited subtree involves mounted information. However, editing of information and "writing through" to remote systems potentially involves significant complexity, particularly if transactions and locking across multiple configuration items are involved. Support for such operations will require additional capabilities, specification of which is beyond the scope of this specification.

Likewise, YANG-Mount does not extend towards RPCs that are defined as part of YANG modules whose contents is being mounted. Support for RPCs that involve mounted portions of the datastore, while conceivable, would require introduction of an additional capability, whose definition is outside the scope of this specification.

By the same token, YANG-Mount does not extend towards notifications. It is conceivable to offer such support in the future using a separate capability, definition of which is once again outside the scope of this specification.

4.4. Other considerations

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

It is expected that some implementations will introduce caching schemes. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Caching is not required for YANG-mount to work - in which case access to mounted information is "on-demand", in which the authoritative data node always gets accessed. Whether to perform caching is a local implementation decision.

When caching is introduced, it can benefit from the ability to subscribe to updates on remote data by remote servers. Some optimizations to facilitate caching support will be discussed in section Section 5.4.

5. Data model structure

5.1. YANG mountpoint extensions

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

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

- The second extension, "subtree", serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression. This extension is used to define mountpoints for Alias-Mount, as well as Peer-Mount.

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

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

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

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

Whenever a mount operation is performed, this condition mountpoint. Whenever a mount operation is performed, this condition MUST be validated by the mount client.

5.2. YANG structure diagrams

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

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

5.3. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

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

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

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

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

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

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

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

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

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

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

5.4. Caching

Under certain circumstances, it can be useful to maintain a cache of remote information. Instead of accessing the remote system, requests are served from a copy that is locally maintained. This is particularly advantageous in cases where data is slow changing, i.e. when there are many more "read" operations than changes to the underlying data node, and in cases when a significant delay were incurred when accessing the remote system, which might be prohibitive for certain applications. Examples of such applications are applications that involve real-time control loops requiring response times that are measured in milliseconds. However, as data nodes that are mounted from an authoritative datastore represent the "golden copy", it is important that any modifications are reflected as soon as they are made.

It is a local implementation decision of mount clients whether to cache information once it has been fetched. However, in order to support more powerful caching schemes, it becomes necessary for the mount server to "push" information proactively. For this purpose, it

is useful for the mount client to subscribe for updates to the mounted information at the mount server. A corresponding mechanism that can be leveraged for this purpose is specified in draft-ietf-netconf-yang-push-05.

Note that caching large mountpoints can be expensive. Therefore limiting the amount of data unnecessarily passed when mounting near the top of a YANG subtree is important. For these reasons, an ability to specify a particular caching strategy in conjunction with mountpoints can be desirable, including the ability to exclude certain nodes and subtrees from caching. According capabilities may be introduced in a future version of this draft.

5.5. Other considerations

5.5.1. Authorization

Access to mounted information is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

5.5.2. Datastore qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

5.5.3. Mount cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

5.5.4. Implementation considerations

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

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

- The mount client acts as a NETCONF client to a remote system. Alternatively, another interface to the remote system can be used, such as a REST API using JSON encodings, as specified in [RFC7951]. --> Either way, to the remote system, the mount client constitutes essentially a client application like any other. The mount client in effect IS a special kind of client application.

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

- The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client's behalf. The communication between mount client and remote mount server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

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

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

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested involving a mountpoint is relayed by the mounting system's infrastructure to the mount client. For example, a request to retrieve information from a datastore leads to an invocation of an internal mount client API when a mount point is reached. The mount client then relays a corresponding operation to the remote datastore.

It subsequently relays the result along with any responses back to the invoking infrastructure, which then merges the result (e.g. a retrieved subtree with the rest of the information that was retrieved) as needed. Relaying the result may involve the need to transpose error response codes in certain corner cases, e.g. when mounted information could not be reached due to loss of connectivity with the remote server, or when a configuration request failed due to validation error.

5.5.5. Modeling best practices

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

For those reasons, as a general best practice, models that make use of mount points SHOULD be defined in a way that minimizes the number of mountpoints required. Finely granular mounts, in which multiple mountpoints are maintained with the same remote system, each containing only very small data subtrees, SHOULD be avoided. For example, lists SHOULD only contain mountpoints when individual list elements are associated with different remote systems. To mount data from lists in remote datastores, a container node that contains all list elements SHOULD be mounted instead of mounting each list element individually. Likewise, instead of having mount points refer to nodes contained underneath choices, a mountpoint should refer to a container of the choice.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "ietf-mount@2017-03-30.yang"
module ietf-mount {
  namespace "urn:ietf:params:xml:ns:yang:ietf-mount";
  prefix mnt;

  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>
```

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

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

        Editor: Alexander Clemm
        <mailto:ludwig@clemm.org>

        Editor: Jan Medved
        <mailto:jmedved@cisco.com>

        Editor: Eric Voit
        <mailto:evoit@cisco.com>";
      description
        "This module provides a set of YANG extensions and definitions
         that can be used to mount information from remote datastores.";

      revision 2017-03-30 {
        description
          "Initial revision.";
        reference
          "draft-clemm-netmod-mount-06.txt";
      }

      extension mountpoint {
        argument name;
        description
          "This YANG extension is used to mount data from another
           subtree in place of the node under which this YANG extension
           statement is used.

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

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

           The extension can take two parameters, target and subtree,
           each defined as their own YANG extensions.

           For Alias-Mount, a mountpoint statement MUST contain a
           subtree statement for the mountpoint definition to be valid.
           For Peer-Mount, a mountpoint statement MUST contain both a
           target and a subtree substatement for the mountpoint
```

```
          definition to be valid.

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

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

          It is possible for the mounted subtree to in turn contain a
          mountpoint.  However, circular mount relationships MUST NOT
          be introduced. For this reason, a mounted subtree MUST NOT
          contain a mountpoint that refers back to the mounting system
          with a mount target that directly or indirectly contains the
          originating mountpoint.";
      }

      extension target {
        argument target-name;
        description
          "This YANG extension is used to perform a Peer-Mount.
           It is used to specify a remote target system from which to
           mount a datastore subtree.  This YANG
           extension takes one argument which specifies the remote
           system. In general, this argument will contain the name of
           a data node that contains the remote system information. It
           is recommended that the reference data node uses the
           mount-target grouping that is defined further below in this
           module.

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

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

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

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

      typedef mount-status {
        type enumeration {
          enum "ok" {
            description
              "Mounted";
          }
          enum "no-target" {
            description
              "The argument of the mountpoint does not define a
               target system";
          }
          enum "no-subtree" {
            description
              "The argument of the mountpoint does not define a
                root of a subtree";
          }
          enum "target-unreachable" {
            description
              "The specified target system is currently
               unreachable";
          }
          enum "mount-failure" {
            description
              "Any other mount failure";
          }
          enum "unmounted" {
            description
              "The specified mountpoint has been unmounted as the
               result of a management operation";
          }
        }
        description
          "This type is used to represent the status of a
           mountpoint.";
      }

      typedef subtree-ref {
```

```
      type string;
      description
        "This string specifies a path to a datanode. It corresponds
         to the path substatement of a leafref type statement.  Its
         syntax needs to conform to the corresponding subset of the
         XPath abbreviated syntax. Contrary to a leafref type,
         subtree-ref allows to refer to a node in a remote datastore.
         Also, a subtree-ref refers only to a single node, not a list
         of nodes.";
    }

    grouping mount-monitor {
      description
        "This grouping contains data nodes that indicate the
         current status of a mount point.";
      leaf mount-status {
        type mount-status;
        config false;
        description
          "Indicates whether a mountpoint has been successfully
           mounted or whether some kind of fault condition is
           present.";
      }
    }

    grouping mount-target {
      description
        "This grouping contains data nodes that can be used to
         identify a remote system from which to mount a datastore
         subtree.";
      container mount-target {
        description
          "A container is used to keep mount target information
           together.";
        choice target-address-type {
          mandatory true;
          description
            "Allows to identify mount target in different ways,
             i.e. using different types of addresses.";
          case IP {
            leaf target-ip {
              type inet:ip-address;
              description
                "IP address identifying the mount target.";
            }
          }
          case URI {
            leaf uri {
```

```
              type inet:uri;
              description
                "URI identifying the mount target";
            }
          }
          case host-name {
            leaf hostname {
              type inet:host;
              description
                "Host name of mount target.";
            }
          }
          case node-ID {
            leaf node-info-ref {
              type subtree-ref;
              description
                "Node identified by named subtree.";
            }
          }
          case other {
            leaf opaque-target-ID {
              type string;
              description
                "Catch-all; could be used also for mounting
                 of data nodes that are local.";
            }
          }
        }
      }
    }

    grouping mount-policies {
      description
        "This grouping contains data nodes that allow to configure
         policies associated with mountpoints.";
      leaf manual-mount {
        type empty;
        description
          "When present, a specified mountpoint is not
           automatically mounted when the mount data node is
           created, but needs to mounted via specific RPC
           invocation.";
      }
      leaf retry-timer {
        type uint16;
        units "seconds";
        description
          "When specified, provides the period after which
```

```
            mounting will be automatically reattempted in case of a
            mount status of an unreachable target";
      }
      leaf number-of-retries {
        type uint8;
        description
          "When specified, provides a limit for the number of
           times for which retries will be automatically
           attempted";
      }
    }

    rpc mount {
      description
        "This RPC allows an application or administrative user to
         perform a mount operation.  If successful, it will result in
         the creation of a new mountpoint.";
      input {
        leaf mountpoint-id {
          type string {
            length "1..32";
          }
          description
            "Identifier for the mountpoint to be created.
             The mountpoint-id needs to be unique;
             if the mountpoint-id of an existing mountpoint is
             chosen, an error is returned.";
        }
      }
      output {
        leaf mount-status {
          type mount-status;
          description
            "Indicates if the mount operation was successful.";
        }
      }
    }
    rpc unmount {
      description
        "This RPC allows an application or administrative user to
         unmount information from a remote datastore.  If successful,
         the corresponding mountpoint will be removed from the
         datastore.";
      input {
        leaf mountpoint-id {
          type string {
            length "1..32";
          }
```

```
          description
            "Identifies the mountpoint to be unmounted.";
        }
      }
      output {
        leaf mount-status {
          type mount-status;
          description
            "Indicates if the unmount operation was successful.";
        }
      }
    }
    container mount-server-mgmt {
      if-feature mount-server-mgmt;
      description
        "Contains information associated with managing the
         mountpoints of a datastore.";
      container mountpoints {
        description
          "Keep the mountpoint information consolidated
           in one place.";
        list mountpoint {
          key "mountpoint-id";
          description
            "There can be multiple mountpoints.
             Each mountpoint is represented by its own
             list element.";
          leaf mountpoint-id {
            type string {
              length "1..32";
            }
            description
              "An identifier of the mountpoint.
               RPC operations refer to the mountpoint
               using this identifier.";
          }
          leaf mountpoint-origin {
            type enumeration {
              enum "client" {
                description
                  "Mountpoint has been supplied and is
                   manually administered by a client";
              }
              enum "auto" {
                description
                  "Mountpoint is automatically
                   administered by the server";
              }
```

```
            }
            config false;
            description
              "This describes how the mountpoint came
               into being.";
          }
          leaf subtree-ref {
            type subtree-ref;
            mandatory true;
            description
              "Identifies the root of the subtree in the
               target system that is to be mounted.";
          }
          uses mount-target;
          uses mount-monitor;
          uses mount-policies;
        }
      }
      container global-mount-policies {
        description
          "Provides mount policies applicable for all mountpoints,
           unless overridden for a specific mountpoint.";
        uses mount-policies;
      }
    }
  }
```

<CODE ENDS>

7. Security Considerations

TBD

8. Acknowledgements

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

9. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <http://www.rfc-editor.org/info/rfc2866>.

[RFC3768] Hinden, R., Ed., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, DOI 10.17487/RFC3768, April 2004, <http://www.rfc-editor.org/info/rfc3768>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[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>.

[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>.

[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>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <http://www.rfc-editor.org/info/rfc7923>.

[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>.

[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>.

[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>.

Appendix A. Example

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

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

The controller network model consists of the following key components:

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

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

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

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

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

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

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

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

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

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

Interface information from NE1 datastore:

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

Interface information from NE2 datastore:

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

NC datastore with mounted interface information from NE1 and NE2:

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

Authors' Addresses

Alexander Clemm
Huawei

EMail: ludwig@clemm.org

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

I2RS working group
Internet-Draft
Intended status: Informational
Expires: September 12, 2017

S. Hares
Huawei
A. Dass
Ericsson
March 11, 2017

Yang for I2RS Protocol
draft-hares-netmod-i2rs-yang-04.txt

Abstract

This document requests yang language additions for the data models that exist as part of the I2RS control plane datastore. One of these additions is the ability to mark a portion of the model as having ephemeral state.

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 12, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
2. Definitions 3
2.1. Requirements language 3
2.2. I2RS Definitions 3
3. yang additions 4
3.1. datastoredef 4
3.2. datastore 6
3.3. dstype 8
3.4. ephemeral 9
3.5. module-list 9
3.6. precedence 9
3.6.1. precedenceval 10
3.7. protosup statement 10
3.7.1. protobase 11
3.7.2. protoadd 11
3.8. validation 12
3.8.1. bulkcheck 13
3.8.2. caching 13
3.8.3. nstransport 14
4. Change to RFC7950 14
4.1. Additions to the Module table 15
4.2. Additions to the submodule substatement list 16
4.3. Additions to the container substatement list 18
4.4. Additions to leaf substatement list 18
4.5. Additions to leaf-list substatement list 19
4.6. Additions to list substatement list 20
4.7. Additions to the grouping substatement table 21
4.8. Additions to the rpc substatement list 22
4.9. Additions to the action substatement list 23
5. IANA Considerations 24
6. Security Considerations 24
7. Acknowledgements 24
8. References 24
8.1. Normative References: 25
8.2. Informative References 25
Authors' Addresses 26

1. Introduction

This a proposal for additions to yang 1.1 [RFC7950] to support the I2RS protocol.

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of IETF existing protocols

(NETCONF [RFC6241], RESTCONF [RFC8044], and others) which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

This document suggests the following additions to Yang to support the I2RS control plane datastore. [I-D.ietf-i2rs-ephemeral-state] specifies the I2RS requirements for the ephemeral state.

Section 3 of this document defines optional additions to yang 1.1 to support I2RS ephemeral control plane datastore. The main addition is the datastore statement with four new substatements (dstype, ephemeral, protosup, validation). The protosup substatement has two valid substatements (protobase, protoadd). The validation substatement has has three new substatements: bulkchecks, caching, and nstransport.

Section 4 provides the augmentation to RFC7950 tables for these optional features.

2. Definitions

2.1. 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].

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended configuration datstore, the dynamic configuration protocols, and control planes datastore to create the applied datastore and operational state datastore.

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration is defined as the intended datastore as defined in [I-D.ietf-netmod-revised-datastores].

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

applied datastore Read only datastore regarding configuration state installed in the routing system as defined in [I-D.ietf-netmod-revised-datastores].

operational state Read only datastore that combines applied datastore and operational state as defined in [I-D.ietf-netmod-revised-datastores].

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with intended configuration (see [I-D.ietf-netmod-revised-datastores]). This operator policy consists of setting a priority for each of the following (per [I-D.ietf-i2rs-ephemeral-state]):

* intended configuration,

* any dynamic configuration protocols,

* any control plane datastores (one of which is ephemeral.)

3. yang additions

3.1. datastoredef

The "datastoredef" is a statement that defines a datastore provides the ability to describe which datastore a module or submodule may be loaded into. Each datastore statement must refer to a name defined in a datastoredef statement.

The new substatements for the datstoredef command are dstype, ephemeral, module-list, precedence, protosup, and validation. The dstype provides information on the type of the datastore. The ephemeral flag indicates the datastore is ephemeral. The module-list provides a list of modules included in this datastore. the protosup indicates the protocol support for this datastore, and the validation provides information on the validation.

The "dsname" must be MUST be a nmae registered with IANA (see [I-D.ietf-netmod-revised-datastores]).

Data store syntax:

```
datastoredef <dsname> {
   <sub-statements>
};
```

dsname - Must be registered name for datastore

Figure 1

The substatements for the datastore substatement are listed below:

Table 1

substatement	This document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
dstype	3.3	-	1
ephemeral	3.4	-	0..n
module		7.1	0..n
module-list	3.5	-	0..n
organization	-	7.1.7	0..1
precedence	3.6	-	0..n
protosup	3.7	-	0..n
reference	-	7.21.4	0..1
revision	-	7.1.9	0..n
revision-date	-	7.1.5.1	0..1
validation	3.8	-	0..n
version		7.1.9	0..n

Note:There is a variance with ephemeral control-plane datastore example in [I-D.ietf-netmod-revised-datastores] which uses "module" to define a datastore rather than datastore. Rather than assume the "module" will be re-used this document uses "datastoredef".

Example of use:

```
datastoredef i2rs-agent {
         dstype control-plane;
     description {"I2RS Agent datastore "};
         ephemeral true;
         module-list ietf-i2rs-rib, ietf-network, ietf-network-topology,
          ietf-l3-unicast-topology;
         protosup {
                protobase netconf;
                protoadd control-plane;
            protoadd ephemeral;
          }
          precedence applied {
           precedenceval 5; //set to high value//
          }
     precedence opstate {
           precedence 5;   //set to high value//
         }
         revision 0.0;
         version 1.0;

}
```

3.2. datastore

The "datastore" can be a substatement for the Yang Module statement provides the ability to describe which datastore a module or submodule may be loaded into. If no "datastore" statement exists, there is no restriction on the datastores a module or submodule can be loaded into.

The argument the datastore is a datastore name denoted as "dsname". The "dsname" must be MUST be a nmae registered with IANA (see [I-D.ietf-netmod-revised-datastores]).

The vaid substatements for the datastore statement are in Table 1. The "description" substatement provides a description of the datastore. The "dstype" provides information on the class (e.g., config or control plane) and the subclass of the datastore (e.g., i2rs). The ephemeral indicates that entire datastore is ephemeral. The validation provides alternate validation rules for the datastore.

Data store syntax:

```
datastore <dsname> {
   <sub-statements>
};
```

dsname - must be defined in a datastoredef statement

Figure 2

The substatements for the datastore substatement are listed below:

Table 2

substatement	This document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
dstype	3.4	-	1
ephemeral	3.5	-	0..n
protosup	3.7	-	0..n
reference	-	7.21.4	0..1
revision	-	7.1.9	0..n
revision-date	-	7.1.5.1	0..1
validation	3.8	-	0..n
version		7.1.9	0..n

Application Comments:

A module may be mounted into different datastores. The datastore statement indicates which datastores a module may be mounted in, and the characteristics of each datastore.

Example of use where a module is utilized
in two different datastores.

```
module ietf-i2rs-rib {
      yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-rib";
     // replace with iana namespace when assigned
         prefix "iir";
         import ietf-inet-types {
            prefix inet;
                //rfc6991
                }
          ....
     organization
       "IETF I2RS (Interface to Routing System) Working Group";
          ....

       datastore i2rs-agent {
             dstype "control-plane" "i2rs-vo";
                ephemeral true;
                protosup {
                 protobase netconf;
                 protoadd control-plane;
                 protoadd ephemeral;
                 }
           }
           datastore config {
                 dstype config;
                 ephemeral false;
                 protosup {
                  protobase restconf;
                 }
                 protosup {
                  protobase netconf;
                  }
           }

}
```

3.3. dstype

They substatement dstype indicates the datastore class and subclass of the datastore. A dstype substatement may only exist within a datastore statement.

Syntax of the dstype datastore is:

```
dstype <dsclass> <dsname>;

where:
    dsclass: ["config" | "control-plane ]
    dssubclass [ "i2rs-v0" ]
```

Figure 3

3.4. ephemeral

The ephemeral indicates that an object is ephemeral data which does not survive a reboot (see [I-D.ietf-i2rs-ephemeral-state]). The definition of the object may be a datastore, a module, a submodule, an action, a container, a grouping, a leaf, a leaf-list, a list, or an rpc.

Syntax is the following:
ephemeral [true | false];

The value "true" indicates the object is not ephemeral.
The value "false" indicates the value is ephemeral.

Figure 4

Note: There is a variance with ephemeral functionality with [I-D.ietf-netmod-revised-datastores]. Rather than consider the keyword ephemeral as a identity, this proposes ephemeral will be a yang substatement.

3.5. module-list

The module list contains a list of module names.

Syntax is the following:
module-list <module-name-1>, ... <module-name-n>;

Each name on the list (e.g. <module-name-1>)
must be a name in a module statement.

Figure 5

3.6. precedence

The precedence provides the value for precedence insertion of the datastores (the precedence substatement is contained) versus the datastore "dsname". Examples of datastore can be applied, opstate,

or other control plane datastores. If no precedence is statement is given, the configuration datastore takes precedence.

The module-list restricts this precedence for just these modules. A submodule must belong to one of the modules in the module list, and it further restricts the precedence value to just the submodule within the module.

Syntax is the following:

```
precedence [applied | opstate | <dsname> ] {
    <<precedence-substatements>>
};
```

dsname - registered name for datastore

Figure 6

Table 3

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
module-list	3.x	-	0..n
precedenceval	3.x	-	1
sub-modules		7.2	0..n

3.6.1. precedenceval

The precedenceval provides the value for precedence.

Syntax is the following:

```
precedenceval <precedence-integer>;
```

<precedence-integer>; - is the integer value for precedence.

Figure 7

3.7. protosup statement

This indicates which protocols support this datastore. The protocols can be netconf, restconf, coap, gprc, and bgp. The substatements for protosup are protocobase and protoadd

```
Syntax is the following:
protosup  {
    <<protosup substatements>>
}
```

Figure 8

Table 4

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
protobase	3.4.1	-	1..n
protoadd	3.4.2	-	1..n

3.7.1. protobase

The protobase substatement indicates the protocol a database can be sent over. The syntax is below:

Syntax for protobase:

```
protobase  [netconf | restconf | coap
            | bgp | isis | ospf | dots
                        | <protocol-name> ]
```

Where protocolo-name is one of protocol names registered by IANA.

Figure 9

3.7.2. protoadd

The protoadd specifies required optional additions to a protocol that sends information to a datastore. One example of such an addition is the additions to RESTCONF to support the I2RS protocol denoted as "i2rs".

Syntax for proto add:

```
protoadd  [control-plane | i2rs | i2nsf |
          | ephemeral | <proto-add-string>]
```

Figure 10

The protocol additions is the name of a capability or grouping of capabilities for support. For example, the "i2rs" capability is a capability which combines the capabilities the "control-plane" netconf capability with the netconf ephemeral capability.

3.8. validation

The validation keyword indicates that the object uses alternate validation besides the mechanisms defined by the configuration datastore as defined in [RFC7950]. The validation subcommand is invalid in any module or submodule which is only defined for the configuration datastore. Unless the module has a datastore statement which includes a datastore other than config, all validation statements in the module are ignored. Unless the submodule has a datastore statement which includes a datastore other than config, all validation statements are ignored.

The validation can be set on a datastore command in a a module, a submodule, an action, a container, a grouping, a leaf, a leaf-list, a list, or an rpc. The validation substatements include nstransport and bulk-checks as shown in table 3.

Syntax of the validation is:

```
 validation {
    <<validation-substatements>>
   };
```

Figure 11

Table 5

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
bulkchecks	3.8.1	-	0..1
caching	3.8.2	-	0..1
nstransport	3.8.3	-	0..1
organization	-	7.1.7	0..1
reference	-	7.21.4	0..n
revision-date	-	7.1.5.1	0..1
version		7.1.9	0..n

3.8.1. bulkcheck

The bulkcheck flag indicates whether this object uses bulk-check validation instead of the normal configuration datastore validation. The protocol updating the object must support bulk checking mechanism, or indicate that this object is not supported.

This is a new feature for control plane protocols and control plane datastores. In the configuration datastores, it is possible to support this feature at the validation level for the rpc object. Early implemementers of this feature for module which may loaded in the configuration datastore are encouraged to place bulkcheck features within "rpc" functionality.

bulkcheck syntax:

```
bulkchecks  [yes | no];
```

Figure 12

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

3.8.2. caching

The caching flag indicates whether the I2RS support caching of multiple client information within I2RS Agents.

Application note: This feature is not supported for the I2RS protocol version 0

caching syntax:

```
caching  [yes | no];
```

Figure 13

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

3.8.3. nstransport

The nstransport indicates whether this object may be sent across a non-secure transport. Sending data across a non-secure transport should be done only if the circumstances warrant it.

This is a new feature for the I2RS control plane protocols and control plane datastores.

Caution: For a description of when a on-secure transport is appropriate for I2RS control plane protocol, please refer to the I2RS protocol security requirements [I-D.ietf-i2rs-protocol-security-requirements]. Implementers of this feature in an I2RS implementation should also review the I2RS security requirements [I-D.ietf-i2rs-security-environment-reqs]. No data which reveals any identity for a person or confidential information should be sent via a non-secure transport.

Syntax is the following:

```
nstransport  [yes | no];
```

Figure 14

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

4. Change to RFC7950

The optional attributes add options to the tables for substatements for the module (section 7.1.1), submodule table, action, container, grouping, leaf, leaf-list, a list, and an rpc. This section provides the revised tables.

4.1. Additions to the Module table

This is the additions to module's substatment table in section 7.1.1 of [RFC7950].

7.1.1 substatement (replacement)

substatement	RFC7950 section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1
optional Yang 1.1 substatement	This document's section	cardinality
datastore	3.2	0..n
ephemeral	3.4	0..n
validation	3.8	0..n

4.2. Additions to the submodule substatement list

Below would be the replacement for the substatement table in setion 7.2.1 of [RFC7950] which lists the valid submodule statements.

7.2.1. The submodule's Substatements (replcement)

substatement	RFC7950 section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
belongs-to	7.2.2	1
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.3. Additions to the container substatement list

Below would be the replacement for the substatement table in section 7.5.2 of [RFC7950] that lists the legal container substatements.

7.5.2. The container Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
notification	7.16	0..n
presennce	7.5.5	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.4. Additions to leaf substatement list

Below would be replacement for the substatement table in section 7.6.2 of [RFC7950] which provides the leaf's substatements.

7.6.2 The leaf's Substatements (replacement)

substatement	RFC7950 section	cardinality
config	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.5. Additions to leaf-list substatement list

Below would be the replacement for the substatement table in section 7.7.2 in [RFC7950] which provides the list of the leaf-lists substatements.

7.7.2 The leaf's Substatements (replacement)

substatement	RFC7950 section	cardinality
config	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1

optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.6. Additions to list substatement list

Below would be the replacement for the table in section 7.8.1 in [RFC7950] which provides the list's substatements.

7.8.1 The list's Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
key	7.8.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
notification	7.16	0..n }
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.7. Additions to the grouping substatement table

Below would be the replacement for the table 7.12.1 of [RFC7950] that lists the vaid substatments for the container substatements.

7.12.1. The grouping's Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
description	7.21.3	0..1
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
uses	7.13	0..n
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.8. Additions to the rpc substatement list

Below would be the replacement for the table in section 7.14.1 of [RFC7950] that lists the legal rpc substatements.

7.5.2. The rpc Substatements

substatement	RFC7950 section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.9. Additions to the action substatement list

Below would be the replacement for the table 7.15.1 of [RFC7950] that lists the legal action substatements.

7.5.2. The action Substatements

substatement	RFC7950 section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
optional Yang 1.1 substatement	This document's section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

Figure 2

5. IANA Considerations

The additions for registries go here.

6. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements]. The security environment the I2RS protocol is covered in [I-D.ietf-i2rs-security-environment-reqs]. Any person implementing or deploying these yang additions for an I2RS protocol should consider both security requirements.

7. Acknowledgements

tBD

8. References

8.1. 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>.

[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>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <http://www.rfc-editor.org/info/rfc7921>.

[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>.

[RFC8044] DeKok, A., "Data Types in RADIUS", RFC 8044, DOI 10.17487/RFC8044, January 2017, <http://www.rfc-editor.org/info/rfc8044>.

8.2. Informative References

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 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-17 (work in progress), September 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-03 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", draft-ietf-netmod-revised-datastores-00 (work in progress), December 2016.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

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

March 13, 2017

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-04

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU. These properties are common to many types of interfaces on network routers and switches and are implemented by multiple network equipment vendors with similar semantics, even though some of the features are not formally defined in any published standard.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Terminology 4
1.2. Tree Diagrams 4
2. Objectives 4
3. Interfaces Common Module 4
3.1. Bandwidth 6
3.2. Carrier Delay 6
3.3. Dampening 7
3.3.1. Suppress Threshold 7
3.3.2. Half-Life Period 8
3.3.3. Reuse Threshold 8
3.3.4. Maximum Suppress Time 8
3.4. Encapsulation 8
3.5. Loopback 8
3.6. MTU 9
3.7. Sub-interface 9
3.8. Forwarding Mode 9
4. Interfaces Ethernet-Like Module 10
5. Interfaces Common YANG Module 10
6. Interfaces Ethernet-Like YANG Module 19
7. Open Issues 23
8. Acknowledgements 23
9. ChangeLog 23
9.1. Version -04 23
9.2. Version -03 23
9.3. Version -02 23
10. IANA Considerations 23
11. Security Considerations 23
11.1. interfaces-common.yang 24
11.2. interfaces-ethernet-like.yang 25
12. References 25
12.1. Normative References 25
12.2. Informative References 26
Authors' Addresses 26

1. Introduction

This document defines two YANG 1.1 [RFC7950] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model [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:

ietf-interfaces-common.yang - Defines extensions to the IETF interface data model to support common configuration data nodes.

ietf-interfaces-ethernet-like.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 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 Common 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 forwarding mode leaf to indicate the OSI layer at which the interface handles traffic
- A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "ietf-interfaces-common" YANG module has the following structure:

```
module: ietf-interfaces-common
  augment /if:interfaces/if:interface:
    +--rw bandwidth?         uint64 {bandwidth}?
    +--rw carrier-delay {carrier-delay}?
    |  +--rw down?   uint32
    |  +--rw up?     uint32
    +--rw dampening! {dampening}?
    |  +--rw half-life?           uint32
    |  +--rw reuse?               uint32
    |  +--rw suppress?            uint32
    |  +--rw max-suppress-time?   uint32
    +--rw encapsulation
    |  +--rw (encaps-type)?
    +--rw loopback?          identityref {loopback}?
    +--rw l2-mtu?            uint16 {configurable-l2-mtu}?
    +--rw forwarding-mode?   identityref {forwarding-mode}?
  augment /if:interfaces/if:interface:
    +--rw parent-interface    if:interface-ref {sub-interfaces}?
```

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

A layer 2 MTU configuration leaf (l2-mtu) is provided to specify the maximum size of a layer 2 frame that may be transmitted or received on an interface. The layer 2 MTU includes the overhead of the layer 2 header and the maximum length of the payload, but excludes any frame check sequence (FCS) bytes. The payload MTU available to higher layer protocols is calculated from the l2-mtu leaf 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.

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. Forwarding Mode

The forwarding mode leaf provides additional information as to what mode or 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 forwarding traffic at 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 "ietf-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
       +--ro statistics
          +--ro in-drop-unknown-dest-mac-pkts?   yang:counter64
```

5. Interfaces Common YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223].

```
<CODE BEGINS> file "ietf-interfaces-common@2017-03-13.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, 2017 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 2017-03-13 {
    description
      "Initial version";
```

```
      reference "Internet draft: draft-ietf-netmod-intf-ext-yang-04";
    }

    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 forwarding-mode {
    description
      "This feature indicates that the device supports the
       configurable forwarding mode leaf";
    reference "Section 3.8 Forwarding Mode";
  }

  /*
   * 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 identity 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";
      }

      identity forwarding-mode {
        description "Base identity for forwarding-mode options";
      }
      identity optical-layer {
        base forwarding-mode;
        description
          "Traffic is being forwarded at the optical layer.  This
           includes DWDM or OTN based switching";
      }
      identity layer-2-forwarding {
        base forwarding-mode;
        description
          "Layer 2 based forwarding, such as Ethernet/VLAN based
           switching, or L2VPN services.";
      }
      identity network-layer {
        base forwarding-mode;
        description
          "Network layer based forwarding, such as IP, MPLS, or L3VPNs";
      }


      /*
       * Augments the IETF interfaces model with a leaf to explicitly
       * specify the bandwidth available on an interface.
       */
      augment "/if:interfaces/if:interface" {
        description
          "Augments the IETF interface model with optional common
           interface level commands that are not formally covered by any
           specific standard";

        leaf bandwidth {
          if-feature "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.
         */
        container carrier-delay {
          if-feature "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
        */
       container dampening {
         if-feature "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.
       */
      container encapsulation {
        when
          "derived-from-or-self(../if:type,
                                'ianaift:ethernetCsmacd') or
           derived-from-or-self(../if:type,
                                'ianaift:ieee8023adLag') or
           derived-from-or-self(../if:type, 'ianaift:pos') or
           derived-from-or-self(../if:type,
                                'ianaift:atmSubInterface') or
           derived-from-or-self(../if:type, 'ethSubInterface')" {

          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
          "Holds the OSI layer 2 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.
        */
      leaf loopback {
        when "derived-from-or-self(../if:type,
                                   'ianaift:ethernetCsmacd') or
```

```
              derived-from-or-self(../if:type, 'ianaift:sonet') or
              derived-from-or-self(../if:type, 'ianaift:atm') or
              derived-from-or-self(../if:type, 'ianaift:otnOtu')" {
            description
              "All interface types that support loopback configuration.";
          }
          if-feature "loopback";
          type identityref {
            base loopback;
          }
          description "Enables traffic loopback.";
        }

        /*
         * Many types of interfaces support a configurable layer 2 MTU.
         */
        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.";
        }

        /*
         * Augments the IETF interfaces model with a leaf that indicates
         * which mode, or layer, is being used to forward the traffic.
         */
        leaf forwarding-mode {
          if-feature "forwarding-mode";
          type identityref {
            base forwarding-mode;
          }

          description
            "The forwarding mode that the interface is operating in";
        }
      }

      /*
       * 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
            derived-from-or-self(if:type, 'ianaift:atmSubInterface') or
            derived-from-or-self(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.";
      }
    }
  }
  <CODE ENDS>
```

6. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223] for Ethernet-like 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@2017-03-13.yang"
module ietf-interfaces-ethernet-like {
  yang-version 1.1;

  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 2017-03-13 {
  description "Updated reference to new internet draft name.";
```

```
      reference
        "Internet draft: draft-ietf-netmod-intf-ext-yang-04";
    }
    /*
     * Configuration parameters for Ethernet-like interfaces.
     */
    augment "/if:interfaces/if:interface" {
      when "derived-from-or-self(if:type, 'ianaift:ethernetCsmacd') or
            derived-from-or-self(if:type, 'ianaift:ieee8023adLag') or
            derived-from-or-self(if:type, 'ianaift:l2vlan') or
            derived-from-or-self(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 Ethernet-like interfaces.
     */
    augment "/if:interfaces-state/if:interface" {
      when "derived-from-or-self(if:type, 'ianaift:ethernetCsmacd') or
            derived-from-or-self(if:type, 'ianaift:ieee8023adLag') or
            derived-from-or-self(if:type, 'ianaift:l2vlan') or
            derived-from-or-self(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.";
        }

        container statistics {
          description
            "Packet statistics that apply to all Ethernet-like
             interfaces";
          leaf in-drop-unknown-dest-mac-pkts {
            type yang:counter64;
            units frames;
            description
              "A count of the number of frames that were well formed,
               but otherwise dropped because the destination MAC
               address did not pass any ingress destination MAC address
               filter.

               For consistency, frames counted against this drop
               counters are also counted against the IETF interfaces
               statistics.  In particular, they are included in
               in-octets and in-discards, but are not included in
               in-unicast-pkts, in-multicast-pkts or in-broadcast-pkts,
               because they are not delivered to a higher layer.

               Discontinuities in the values of this counters in this
               container can occur at re-initialization of the
               management system, and at other times as indicated by
               the value of the 'discontinuity-time' leaf defined in
               the ietf-interfaces YANG module (RFC 7223).";
          }
        }
      }
    }
  }
  <CODE ENDS>
```

7. Open Issues

Open issues:

1. Should the loopback leaf be extended to also cover features such as dataplane loopback?

2. Does loopback need a action statement to allow it to be enabled in an ephemeral way (specifically lost on reboot)?

3. Should the bandwidth leaf be renamed 'reported-bandwidth'? Should this leaf be defined in a routing YANG module?

8. Acknowledgements

The authors wish to thank Eric Gray, Ing-Wher Chen, Juergen Schoenwaelder, Ladislav Lhotka, Mahesh Jethanandani, Michael Zitao, Neil Ketley, Qin Wu, William Lupton, and Xufeng Liu for their helpful comments contributing to this draft.

9. ChangeLog

9.1. Version -04

- Incorporate feedback from Lada, some comments left as open issues.

9.2. Version -03

- Fixed incorrect module name references, and updated tree output

9.3. Version -02

- Minor changes only: Fix errors in when statements, use derived-from-or-self() for future proofing.

10. 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".

11. 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:

11.1. 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/forwarding-mode

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

11.2. 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

12. References

12.1. 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>.

[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>.

[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>.

12.2. 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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

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

July 2, 2018

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-06

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU. These properties are common to many types of interfaces on network routers and switches and are implemented by multiple network equipment vendors with similar semantics, even though some of the features are not formally defined in any published standard.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Terminology 4
1.2. Tree Diagrams 4
2. Objectives 4
3. Interfaces Common Module 4
3.1. Carrier Delay 6
3.2. Dampening 7
3.2.1. Suppress Threshold 8
3.2.2. Half-Life Period 8
3.2.3. Reuse Threshold 8
3.2.4. Maximum Suppress Time 8
3.3. Encapsulation 8
3.4. Loopback 9
3.5. Layer 2 MTU 9
3.6. Sub-interface 9
3.7. Forwarding Mode 10
4. Interfaces Ethernet-Like Module 11
5. Interfaces Common YANG Module 11
6. Interfaces Ethernet-Like YANG Module 22
7. Examples 25
7.1. Carrier delay configuration 25
7.2. Dampening configuration 26
7.3. MAC address configuration 27
8. Acknowledgements 28
9. ChangeLog 29
9.1. Version -06 29
9.2. Version -05 29
9.3. Version -04 29
9.4. Version -03 29
9.5. Version -02 29
10. IANA Considerations 29
11. Security Considerations 29
11.1. interfaces-common.yang 30
11.2. interfaces-ethernet-like.yang 31
12. References 31
12.1. Normative References 31
12.2. Informative References 32
Authors' Addresses 32

1. Introduction

This document defines two NMDA compatible [RFC8342] YANG 1.1 [RFC7950] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model [RFC8343] 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:

ietf-interfaces-common.yang - Defines extensions to the IETF interface data model to support common configuration data nodes.

ietf-interfaces-ethernet-like.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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 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 Common Module

The Interfaces Common module provides some basic extensions to the IETF interfaces YANG module.

The module provides:

- 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 forwarding mode leaf to indicate the OSI layer at which the interface handles traffic
- A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "ietf-interfaces-common" YANG module has the following structure:

```
module: ietf-interfaces-common
  augment /if:interfaces/if:interface:
    +--rw carrier-delay {carrier-delay}?
    |  +--rw down?                   uint32
    |  +--rw up?                     uint32
    |  +--ro carrier-transitions?    yang:counter64
    |  +--ro timer-running?          enumeration
    +--rw dampening! {dampening}?
    |  +--rw half-life?            uint32
    |  +--rw reuse?                uint32
    |  +--rw suppress?             uint32
    |  +--rw max-suppress-time?    uint32
    |  +--ro penalty?              uint32
    |  +--ro suppressed?           boolean
    |  +--ro time-remaining?       uint32
    +--rw encapsulation
    |  +--rw (encaps-type)?
    +--rw loopback?           identityref {loopback}?
    +--rw l2-mtu?             uint16 {configurable-l2-mtu}?
    +--rw forwarding-mode?    identityref {forwarding-mode}?
  augment /if:interfaces/if:interface:
    +--rw parent-interface    if:interface-ref {sub-interfaces}?
```

3.1. 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 3.2 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 may occur. The default value for this leaf is determined by the underlying network device.

3.2. 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.

3.2.1. 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.

3.2.2. 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.

3.2.3. 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.

3.2.4. 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.

3.3. 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. The use of a choice statement ensures that an interface can only have a single datalink layer protocol configured.

The different encapsulations themselves are defined in separate YANG modules defined in other documents that augument the encapsulation choice statement. For example the Ethernet specific basic 'dot1q-vlan' encapsulation is defined in ietf-if-l3-vlan.yang and the

'flexible' encapsulation is defined in ietf-flexible-encapsulation.yang, both modules from [I-D.ietf-netmod-sub-intf-vlan-model].

3.4. 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.

The following loopback modes are defined:

- Internal loopback - All egress traffic on the interface is internally looped back within the interface to be received on the ingress path.
- Line loopback - All ingress traffic received on the interface is internally looped back within the interface to the egress path.
- Loopback Connector - The interface has a physical loopback connector attached that loops all egress traffic back into the interface's ingress path, with equivalent semantics to internal loopback.

3.5. Layer 2 MTU

A layer 2 MTU configuration leaf (l2-mtu) is provided to specify the maximum size of a layer 2 frame that may be transmitted or received on an interface. The layer 2 MTU includes the overhead of the layer 2 header and the maximum length of the payload, but excludes any frame check sequence (FCS) bytes. The payload MTU available to higher layer protocols is calculated from the l2-mtu leaf 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.

3.6. 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 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.

3.7. Forwarding Mode

The forwarding mode leaf provides additional information as to what mode or 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.

YANG Modules can conditionally use this leaf as a simple mechanism to determine whether particular types of configuration are valid. YANG modules can write 'must' statements to check whether the forwarding mode leaf has been configured, and if it is, then validate that the specified configuration is consistent with any forwarding mode that has also been configured. E.g., a layer 2 QoS policy YANG module could ensure that it is only applied to a interface forwarding traffic at layer 2 by checking whether the forwarding-mode leaf exists, and if it does then also ensure that it has been set to 'layer-2-forwarding'.

The following forwarding modes are defined:

- Optical Layer - Traffic is being forwarded at the optical layer. This includes DWDM or OTN based switching.

- Layer 2 - Layer 2 based forwarding, such as Ethernet/VLAN based switching, or L2VPN services.

- Network Layer - Network layer based forwarding, such as IP, MPLS, or L3VPNs.

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 "ietf-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
       +--ro bia-mac-address?   yang:mac-address
       +--ro statistics
          +--ro in-drop-unknown-dest-mac-pkts?   yang:counter64
```

5. Interfaces Common YANG Module

This YANG module augments the interface container defined in RFC 8343 [RFC8343].

```
<CODE BEGINS> file "ietf-interfaces-common@2018-07-02.yang"
module ietf-interfaces-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces-common";

  prefix if-cmn;

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

```
   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: Joel Jaeggli
                <mailto:joelja@gmail.com>

      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) 2016-2018 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 2018-07-02 {
     description
       "Initial version";
```

```
    reference "Internet draft: draft-ietf-netmod-intf-ext-yang-06";
  }

  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 "RFC XXX, Section 3.1 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 "RFC XXX, Section 3.2 Dampening";
  }

  feature loopback {
    description
      "This feature indicates that configurable interface loopback
       is supported.";
    reference "RFC XXX, Section 3.4 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 "RFC XXX, Section 3.5 Layer 2 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 "RFC XXX, Section 3.6 Sub-interface";
```

```
   }

   feature forwarding-mode {
     description
       "This feature indicates that the device supports the
        configurable forwarding mode leaf";
     reference "RFC XXX, Section 3.7 Forwarding Mode";
   }

   /*
    * 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";
     reference "RFC XXX, Section 3.6 Sub-interface";
   }

   identity ethSubInterface{
     base ianaift:l2vlan;
     base sub-interface;

     description
       "This identity represents the child sub-interface of any
        interface types that uses Ethernet framing (with or without
        802.1Q tagging)";
   }

   identity loopback {
     description "Base identity for interface loopback options";
     reference "RFC XXX, section 3.4";
   }
   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.";
     reference "RFC XXX, section 3.4";
   }
   identity loopback-line {
     base loopback;
     description
       "All ingress traffic received on the interface is internally
        looped back within the interface to the egress path.";
```

```
    reference "RFC XXX, section 3.4";
  }
  identity loopback-connector {
    base loopback;
    description
      "The interface has a physical loopback connector attached
       that loops all egress traffic back into the interface's
       ingress path, with equivalent semantics to loopback-internal";
    reference "RFC XXX, section 3.4";
  }

  identity forwarding-mode {
    description "Base identity for forwarding-mode options.";
    reference "RFC XXX, section 3.7";
  }
  identity optical-layer {
    base forwarding-mode;
    description
      "Traffic is being forwarded at the optical layer.  This
       includes DWDM or OTN based switching.";
    reference "RFC XXX, section 3.7";
  }
  identity layer-2-forwarding {
    base forwarding-mode;
    description
      "Layer 2 based forwarding, such as Ethernet/VLAN based
       switching, or L2VPN services.";
    reference "RFC XXX, section 3.7";
  }
  identity network-layer {
    base forwarding-mode;
    description
      "Network layer based forwarding, such as IP, MPLS, or L3VPNs.";
    reference "RFC XXX, section 3.7";
  }


  /*
   * Augments the IETF interfaces model with leaves to configure
   * and monitor carrier-delay on an interface.
   */
  augment "/if:interfaces/if:interface" {
    description
      "Augments the IETF interface model with optional common
       interface level commands that are not formally covered by any
       specific standard.";

    /*
```

```
     * Defines standard YANG for the Carrier Delay feature.
     */
    container carrier-delay {
      if-feature "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.";
      }
      leaf carrier-transitions {
        type yang:counter64;
        units transitions;
        config false;
        description
          "Defines the number of times the underlying carrier state
           has changed to, or from, state up.  This counter should be
           incremented even if the high layer interface state changes
           are being suppressed by a running carrier-delay timer.";
      }
      leaf timer-running {
        type enumeration {
          enum none {
            description
```

```
              "No carrier delay timer is running.";
          }
          enum up {
            description
              "Carrier-delay up timer is running.  The underlying
               carrier state is up, but interface state is not
               reported as up.";
          }
          enum down {
            description
              "Carrier-delay down timer is running.  Interface state
               is reported as up, but the underlying carrier state is
               actually down.";
          }
        }
        default "none";
        config false;
        description
          "Reports whether a carrier delay timer is actively running,
           in which case the interface state does not match the
           underlying carrier state.";
      }

      reference "RFC XXX, Section 3.1 Carrier Delay";
    }

    /*
     * Augments the IETF interfaces model with a container to hold
     * generic interface dampening
     */
    container dampening {
      if-feature "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";
      reference "RFC XXX, Section 3.2 Dampening";
      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.";
        reference "RFC XXX, Section 3.3.2 Half-Life Period";
      }
      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.";
        reference "RFC XXX, Section 3.2.3 Reuse Threshold";
      }

      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.";
        reference "RFC XXX, Section 3.2.1 Suppress Threshold";
      }

      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.";
        reference "RFC XXX, Section 3.2.4 Maximum Suppress Time";
      }

      leaf penalty {
        type uint32;
        config false;
        description
          "The current penalty value for this interface.  When the
           penalty value exceeds the 'suppress' leaf then the
           interface is suppressed (i.e. held down).";
        reference "RFC XXX, Section 3.2 Dampening";
      }

      leaf suppressed {
        type boolean;
        default "false";
```

```
        config false;
        description
          "Represents whether the interface is suppressed (i.e. held
           down) because the 'penalty' leaf value exceeds the
           'suppress' leaf.";
        reference "RFC XXX, Section 3.2 Dampening";
      }

      leaf time-remaining {
        when '../suppressed = "true"' {
          description
            "Only suppressed interfaces should have a time remaining.";
        }
        type uint32;
        units seconds;
        config false;
        description
          "For a suppressed interface, this leaf represents how long
           (in seconds) that the interface will remain suppressed
           before it is allowed to go back up again.";
        reference "RFC XXX, Section 3.2 Dampening";
      }
    }

    /*
     * 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.
     */
    container encapsulation {
      when
        "derived-from-or-self(../if:type,
                              'ianaift:ethernetCsmacd') or
         derived-from-or-self(../if:type,
                              'ianaift:ieee8023adLag') or
         derived-from-or-self(../if:type, 'ianaift:pos') or
         derived-from-or-self(../if:type,
                              'ianaift:atmSubInterface') or
         derived-from-or-self(../if:type, 'ethSubInterface')" {

        description
          "All interface types that can have a configurable L2
           encapsulation";
      }
```

```
      description
        "Holds the OSI layer 2 encapsulation associated with an
         interface";
      choice encaps-type {
        description
          "Extensible choice of layer 2 encapsulations";
        reference "RFC XXX, Section 3.3 Encapsulation";
      }
    }

     /*
      * Various types of interfaces support loopback configuration,
      * any that are supported by YANG should be listed here.
      */
    leaf loopback {
      when "derived-from-or-self(../if:type,
                                 'ianaift:ethernetCsmacd') or
            derived-from-or-self(../if:type, 'ianaift:sonet') or
            derived-from-or-self(../if:type, 'ianaift:atm') or
            derived-from-or-self(../if:type, 'ianaift:otnOtu')" {
        description
          "All interface types that support loopback configuration.";
      }
      if-feature "loopback";
      type identityref {
        base loopback;
      }
      description "Enables traffic loopback.";
      reference "RFC XXX, Section 3.4 Loopback";
    }

    /*
     * Many types of interfaces support a configurable layer 2 MTU.
     */
    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.";
      reference "RFC XXX, Section 3.5 Layer 2 MTU";
    }
```

```
     /*
      * Augments the IETF interfaces model with a leaf that indicates
      * which mode, or layer, is being used to forward the traffic.
      */
     leaf forwarding-mode {
       if-feature "forwarding-mode";
       type identityref {
         base forwarding-mode;
       }

       description
         "The forwarding mode that the interface is operating in.";
       reference "RFC XXX, Section 3.7 Forwarding Mode";
     }
   }

   /*
    * 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
           derived-from-or-self(if:type, 'ianaift:atmSubInterface') or
           derived-from-or-self(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.";
       reference "RFC XXX, Section 3.6 Sub-interface";
     }
   }
 }
```

<CODE ENDS>

6. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 8343 [RFC8343] for Ethernet-like 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@2017-07-03.yang"
module ietf-interfaces-ethernet-like {
  yang-version 1.1;

  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 2017-07-03 {
    description "Updated to conform to NMDA architecture";

    reference
      "Internet draft: draft-ietf-netmod-intf-ext-yang-05";
  }

  /*
   * Configuration parameters for Ethernet-like interfaces.
   */
  augment "/if:interfaces/if:interface" {
    when "derived-from-or-self(if:type, 'ianaift:ethernetCsmacd') or
          derived-from-or-self(if:type, 'ianaift:ieee8023adLag') or
          derived-from-or-self(if:type, 'ianaift:l2vlan') or
          derived-from-or-self(if:type, 'ianaift:ifPwType')" {
      description "Applies to all Ethernet-like interfaces";
    }
    description
      "Augment the interface model with parameters for all
       Ethernet-like interfaces";

    container ethernet-like {
      description
        "Contains parameters for interfaces that use Ethernet framing
         and expose an Ethernet MAC layer.";
      leaf mac-address {
        type yang:mac-address;
        description
```

```
            "The MAC address of the interface.";
          }

          leaf bia-mac-address {
            type yang:mac-address;
            config false;
            description
              "The 'burnt-in' MAC address.  I.e the default MAC address
               assigned to the interface if no MAC address has been
               explicitly configured on it.";
          }

          container statistics {
            config false;
            description
              "Packet statistics that apply to all Ethernet-like
               interfaces";
            leaf in-drop-unknown-dest-mac-pkts {
              type yang:counter64;
              units frames;
              description
                "A count of the number of frames that were well formed,
                 but otherwise dropped because the destination MAC
                 address did not pass any ingress destination MAC address
                 filter.

                 For consistency, frames counted against this drop
                 counters are also counted against the IETF interfaces
                 statistics.  In particular, they are included in
                 in-octets and in-discards, but are not included in
                 in-unicast-pkts, in-multicast-pkts or in-broadcast-pkts,
                 because they are not delivered to a higher layer.

                 Discontinuities in the values of this counters in this
                 container can occur at re-initialization of the
                 management system, and at other times as indicated by
                 the value of the 'discontinuity-time' leaf defined in
                 the ietf-interfaces YANG module (RFC 8343).";
            }
          }
        }
      }
    }
    <CODE ENDS>
```

7. Examples

The following sections give some examples of how different parts of the YANG modules could be used. Examples are not given for the more trivial configuration, or for sub-interfaces, for which examples are contained in [I-D.ietf-netmod-sub-intf-vlan-model].

7.1. Carrier delay configuration

The following example shows how the operational state datastore could look like for an Ethernet interface without any carrier delay configuration. The down leaf value of 0 indicates that link down events as always propagated to high layers immediately, but an up leaf value of 50 indicates that the interface must be up and stable for at least 50 msecs before the interface is reported as being up to the high layers.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
 xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
 xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
xmlns:if-cmn="urn:ietf:params:xml:ns:yang:ietf-interfaces-common">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <if-cmn:carrier-delay>
      <if-cmn:down>0</if-cmn:down>
      <if-cmn:up>1000</if-cmn:up>
    </if-cmn:carrier-delay>
  </interface>
</interfaces>
```

The following example shows explicit carrier delay up and down values have been configured. A 50 msec down leaf value has been used to potentially allow optical protection to recover the link before the higher layer protocol state is flapped. A 1 second (1000 milliseconds) up leaf value has been used to ensure that the link is always reasonably stable before allowing traffic to be carried over it. This also has the benefit of greatly reducing the rate at which higher layer protocol state flaps could occur.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:if-cmn="urn:ietf:params:xml:ns:yang:ietf-interfaces-common">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
      <if-cmn:carrier-delay>
        <if-cmn:down>50</if-cmn:down>
        <if-cmn:up>1000</if-cmn:up>
      </if-cmn:carrier-delay>
    </interface>
  </interfaces>
</config>
```

7.2. Dampening configuration

The following example shows what the operational state datastore may look like for an interface configured with interface dampening. The 'suppressed' leaf indicates that the interface is currently suppressed (i.e. down) because the 'penalty' is greater than the 'suppress' leaf threshold. The 'time-remaining' leaf indicates that the interface will remain suppressed for another 103 seconds before the 'penalty' is below the 'reuse' leaf value and the interface is allowed to go back up again.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
 xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
 xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <dampening
     xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces-common">
      <half-life>60</half-life>
      <reuse>750</reuse>
      <suppress>2000</suppress>
      <max-suppress-time>240</max-suppress-time>
      <penalty>2480</penalty>
      <suppressed>true</suppressed>
      <time-remaining>103</time-remaining>
    </dampening>
  </interface>
</interfaces>
```

7.3. MAC address configuration

The following example shows how the operational state datastore could look like for an Ethernet interface without an explicit MAC address configured. The mac-address leaf always reports the actual operational MAC address that is in use. The bia-mac-address leaf always reports the default MAC address assigned to the hardware.

```
<?xml version="1.0" encoding="utf-8"?>
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
      <ethernet-like
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like">
        <mac-address>00:00:5E:00:53:30</mac-address>
        <bia-mac-address>00:00:5E:00:53:30</bia-mac-address>
      </ethernet-like>
    </interface>
  </interfaces>
```

The following example shows an explicit MAC address being configured on interface eth0.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
      <ethernet-like
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like">
        <mac-address>00:00:5E:00:53:35</mac-address>
      </ethernet-like>
    </interface>
  </interfaces>
</config>
```

After the MAC address configuration has been successfully applied, the operational state datastore reporting the interface MAC address properties would contain the following, with the mac-address leaf updated to match the configured value, but the bia-mac-address leaf retaining the same value - which should never change.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
 xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
 xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <ethernet-like
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like">
      <mac-address>00:00:5E:00:53:35</mac-address>
      <bia-mac-address>00:00:5E:00:53:30</bia-mac-address>
    </ethernet-like>
  </interface>
</interfaces>
```

8. Acknowledgements

The authors wish to thank Eric Gray, Ing-Wher Chen, Juergen Schoenwaelder, Ladislav Lhotka, Mahesh Jethanandani, Michael Zitao, Neil Ketley, Qin Wu, William Lupton, Xufeng Liu, and Andy Bierman for their helpful comments contributing to this draft.

9. ChangeLog

XXX, RFC Editor, please delete this change log before publication.

9.1. Version -06

- Remove reservable-bandwidth, based on Acee's suggestion
- Add examples
- Add additional state parameters for carrier-delay and dampening

9.2. Version -05

- Incorporate feedback from Andy Bierman

9.3. Version -04

- Incorporate feedback from Lada, some comments left as open issues.

9.4. Version -03

- Fixed incorrect module name references, and updated tree output

9.5. Version -02

- Minor changes only: Fix errors in when statements, use derived-from-or-self() for future proofing.

10. 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".

11. 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:

11.1. 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 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/forwarding-mode

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

11.2. 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

12. References

12.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://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>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

12.2. Informative References

[I-D.ietf-netmod-sub-intf-vlan-model]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Sub-interface VLAN YANG Data Models", draft-ietf-netmod-sub-intf-vlan-model-03 (work in progress), October 2017.

[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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

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
Intended status: Standards Track
Expires: September 14, 2017

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
March 13, 2017

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-01

Abstract

Datastores are a fundamental concept binding the data models written in the YANG data modeling language to network management protocols such as NETCONF and RESTCONF. This document defines an architectural framework for datastores based on the experience gained with the initial simpler model, 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 September 14, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction 3
2. Terminology 3
3. Introduction 5
3.1. Original Model of Datastores 7
4. Architectural Model of Datastores 8
4.1. The <intended> Datastore 9
4.2. Dynamic Datastores 10
4.3. The <operational> Datastore 10
4.3.1. Missing Resources 11
4.3.2. System-controlled Resources 11
4.3.3. Origin Metadata Annotation 11
5. Guidelines for Defining Dynamic Datastores 12
5.1. Define a name for the dynamic datastore 12
5.2. Define which YANG modules can be used in the datastore . 12
5.3. Define which subset of YANG-modeled data applies 13
5.4. Define how dynamic data is actualized 13
5.5. Define which protocols can be used 13
5.6. Define a module for the dynamic datastore 13
6. YANG Modules 14
7. IANA Considerations 18
7.1. Updates to the IETF XML Registry 18
7.2. Updates to the YANG Module Names Registry 19
8. Security Considerations 19
9. Acknowledgments 19
10. References 20
10.1. Normative References 20
10.2. Informative References 21
Appendix A. Example Data 22
A.1. System Example 22
A.2. BGP Example 25
A.2.1. Datastores 27
A.2.2. Adding a Peer 27
A.2.3. Removing a Peer 28
A.3. Interface Example 29
A.3.1. Pre-provisioned Interfaces 29
A.3.2. System-provided Interface 30
Appendix B. Ephemeral Dynamic Datastore Example 31
Appendix C. Implications on Data Models 32

C.1. Proposed migration of existing YANG Data Models 33
C.2. Standardization of new YANG Data Models 34
Appendix D. Implications on other Documents 34
D.1. Implications on YANG 34
D.2. Implications on YANG Library 34
D.3. Implications to YANG Guidelines 35
D.3.1. Nodes with different config/state value sets 35
D.3.2. Auto-configured or Auto-negotiated Values 35
D.4. Implications on NETCONF 35
D.4.1. Introduction 36
D.4.2. Overview of additions to NETCONF 36
D.4.3. Overview of NETCONF version 2 37
D.5. Implications on RESTCONF 40
D.5.1. Introduction 40
D.5.2. Overview of additions to RESTCONF 40
D.5.3. Overview of a possible new RESTCONF version 42
Appendix E. Open Issues 43
Authors' Addresses 44

1. Introduction

This document provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols. 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. This architecture is agnostic with regard to the encoding used by network management protocols.

2. 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].

This document defines the following terms:

- configuration data: Data that determines how a device behaves. This data is modeled in YANG using "config true" nodes. Configuration data can originate from different sources.

- static configuration data: Configuration data that is eventually persistent and used to get a device from its initial default state into its desired operational state.

- dynamic configuration data: Configuration data that is obtained dynamically during the operation of a device through interaction with other systems and not persistent.

- system configuration data: Configuration data that is supplied by the device itself.

- default configuration data: Configuration data that is not explicitly provided but for which a value defined in the data model is used.

- applied configuration data: Configuration data that is currently used by a device. Applied configuration data consists of static configuration data and dynamic configuration data.

- state data: The additional data on a system that is not configuration data such as read-only status information and collected statistics. State data is transient and modified by interactions with internal components or other systems. State data is modeled in YANG using "config false" nodes.

- 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. A datastore maps to an instantiated YANG data tree.

- configuration datastore: A datastore holding static configuration data that is required to get a device from its initial default state into a desired operational state. A configuration datastore maps to an instantiated YANG data tree consisting of configuration data nodes and interior data nodes.

- running configuration datastore: A configuration datastore holding the complete static configuration currently active on the device. The running configuration datastore always exists. It may include inactive configuration or template-mechanism-oriented configuration that require further expansion.

- intended configuration datastore: A configuration datastore holding the complete configuration currently active on the device. It does not include inactive configuration and it does include the expansion of any template mechanisms.

- candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's running configuration datastore and that can be committed to the running configuration datastore. A candidate datastore may not be supported by all protocols or implementations.

- startup configuration datastore: The configuration datastore holding the configuration loaded by the device into the running configuration datastore when it boots. A startup datastore may not be supported by all protocols or implementations.

- dynamic datastore: A datastore holding dynamic configuration data.

- operational state datastore: A datastore holding the currently active applied configuration data as well as the device's state data.

- origin: A metadata annotation indicating the origin of a data item.

- remnant data: Configuration data that remains in the system for a period of time after it has be removed from a configuration datastore. The time period may be minimal, or may last until all resources used by the newly-deleted configuration data (e.g., network connections, memory allocations, file handles) have been deallocated.

The following additional terms are not datastore specific but commonly used and thus defined here as well:

- client: An entity that can access YANG-defined data on a server, over some network management protocol.

- server: An entity that provides access to YANG-defined data to a client, over some network management protocol.

- notification: A server-initiated message indicating that a certain event has been recognized by the server.

- remote procedure call: An operation that can be invoked by a client on a server.

3. Introduction

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 [RFC7950] 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.

[RFC6244] 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.

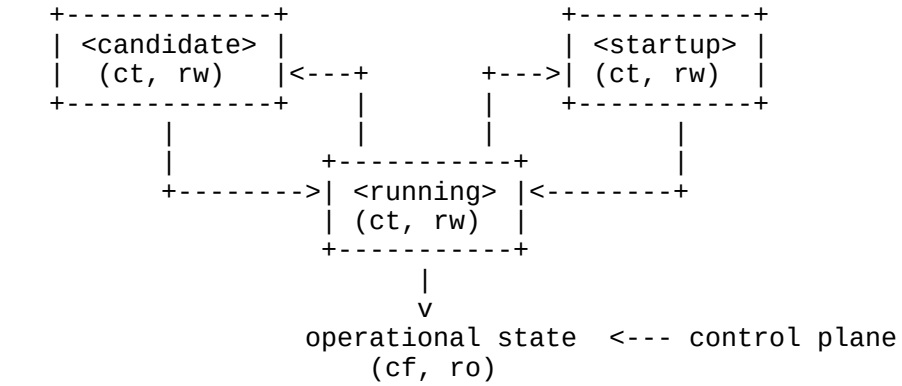
Section 4.3.3 of [RFC6244] discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as a separate data tree is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] 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 in the data model has been found operationally complicated, and typically impacts the readability of module definitions due to overuse of groupings. The relationship between the branches is not machine readable and filter expressions operating on configuration data and on related operational state data are different.

3.1. 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
boxes denote datastores

Note that this diagram simplifies the model: 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 if the operational state data can

have a different lifetime compared to configuration data or if configuration data is not immediately or successfully applied.

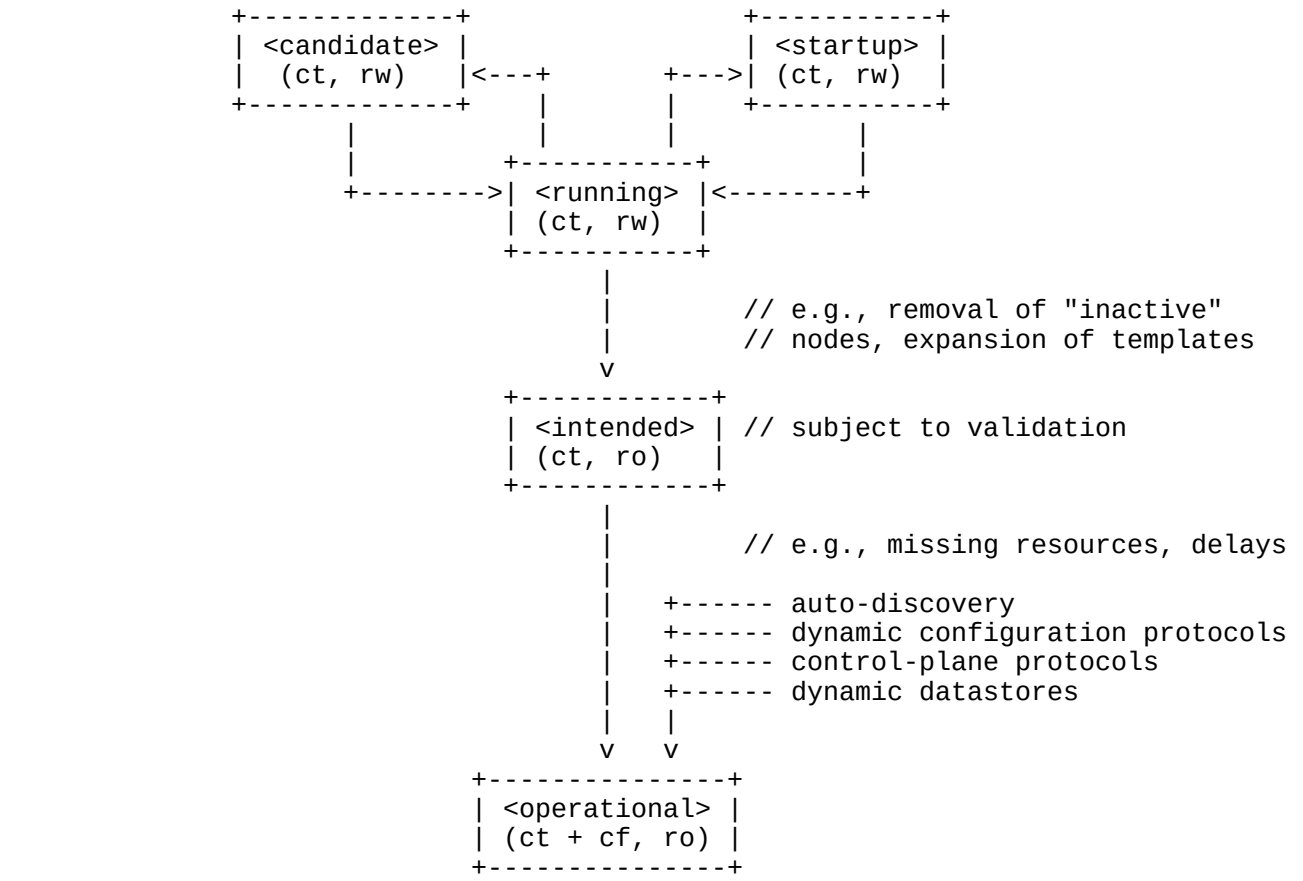
- Several implementations have proprietary mechanisms that allow clients to store inactive data in the <running> datastore; this inactive data is only exposed to clients that indicate that they support the concept of inactive data; clients not indicating support for inactive data receive the content of the <running> datastore with the inactive data removed. Inactive data is conceptually removed before validation.

- Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.

- 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. Architectural Model of Datastores

Below is a new conceptual model of datastores extending the original model in order to reflect the experience gained with the original model.



ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote datastores

4.1. The <intended> Datastore

The <intended> datastore is a read-only datastore that consists of config true nodes. It is tightly coupled to <running>. When data is written to <running>, the data that is to be validated is also conceptually written to <intended>. Validation is performed on the contents of <intended>.

On a traditional NETCONF implementation, <running> and <intended> are always the same.

Currently there are no standard mechanisms defined that affect <intended> so that it would have different contents than <running>, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>, and are thus not taken into account when validating the configuration.

Another example is support for templates. Templates are expanded when copied into <intended>, and the expanded result is validated.

4.2. Dynamic Datastores

The model recognizes the need for dynamic 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 dynamic datastores interact with the rest of the system through the <operational> datastore.

Note that the ephemeral datastore discussed in I2RS documents maps to a dynamic datastore in the datastore model described here.

4.3. The <operational> Datastore

The <operational> datastore is a read-only datastore that consists of config true and config false nodes. In the original NETCONF 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 <operational> datastore contains all configuration data actually used by the system, including all applied configuration, system-provided configuration and values defined by any supported data models. In addition, the <operational> datastore also contains state data.

Changes to configuration data may take time to percolate through to the <operational> datastore. During this period, the <operational> datastore will return data nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device. These "remnants" of the previous configuration persist while the system has released resources used by the newly-deleted configuration data (e.g., network connections, memory allocations, file handles).

As a result of these remnants, the semantic constraints defined in the data model cannot be relied upon for the <operational> datastore, since the system may have remnants whose constraints were valid with the previous configuration and that are not valid with the current configuration. Since constraints on "config false" nodes may refer to "config true" nodes, remnants may force the violation of those constraints. The constraints that may not hold include "when", "must", "min-elements", and "max-elements". Note that syntactic constraints cannot be violated, including hierarchical organization, identifiers, and type-based constraints.

4.3.1. Missing Resources

The <intended> configuration can refer to resources that are not available or otherwise not physically present. In these situations, these parts of the <intended> configuration are not applied. The data appears in <intended> but does not appear in <operational>.

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 <operational>.

Note that configuration validity cannot depend on the current state of such resources, since that would imply the removing a resource might render the configuration invalid. This is unacceptable, especially given that rebooting such a device would fail to boot due to an invalid configuration. Instead we allow configuration for missing resources to exist in <running> and <intended>, but it will not appear in <operational>.

4.3.2. System-controlled Resources

Sometimes resources are controlled by the device and the corresponding system controlled data appear in (and disappear from) <operational> 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 <operational> eventually (if application of the configuration was successful).

4.3.3. Origin Metadata Annotation

As data flows into the <operational> datastore, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The "origin" metadata annotation is defined in Section 6. The values are YANG identities. The following identities are defined:

```
     +-- origin
         +-- static
         +-- dynamic
         +-- default
         +-- system
```

These identities can be further refined, e.g., there might be an identity "dhcp" derived from "dynamic".

The "static" origin represents data provided by the <intended> datastore. The "dynamic" origin represents data provided by a dynamic datastore. The "default" origin represents data values specified in the data model, using either simple values in the "default" statement or any values described in the "description" statement. Finally, the "system" origin represents data learned from the normal operational of the system, including control-plane protocols.

5. Guidelines for Defining Dynamic Datastores

The definition of a dynamic datastore SHOULD be provided in a document (e.g., an RFC) purposed to the definition of the dynamic datastore. When it makes sense, more than one dynamic datastore MAY be defined in the same document (e.g., when the datastores are logically connected). Each dynamic datastore's definition SHOULD address the points specified in the sections below.

5.1. Define a name for the dynamic datastore

Each dynamic datastores MUST have a name using the character set described by Section 6.2 of [RFC7950]. The name SHOULD be consistent in style and length to other datastore names described in this document.

The datastore's name does not need to be globally unique, as it will be uniquely qualified by the namespace of the module in which it is defined (Section 5.6). This means that names such as "running" and "operational" are valid datastore names. However, it is usually desirable to avoid using the same name for multiple different datastores.

5.2. Define which YANG modules can be used in the datastore

Not all YANG modules may be used in all datastores. Some datastores may constrain which data models can be used in them. If it is desirable that a subset of all modules can be targeted to the dynamic datastore, then the documentation defining the dynamic datastore MUST use the mechanisms described in Appendix D.2 to provide the necessary

hooks for module-designers to indicate that their module is to be accessible in the dynamic datastore.

5.3. Define which subset of YANG-modeled data applies

By default, the data in a dynamic datastore is modeled by all YANG statements in the available YANG modules. However, it is possible to specify criteria YANG statements must satisfy in order to be present in a dynamic datastore. For instance, maybe only config true nodes are present, or config false nodes that also have a specific YANG extension (e.g., i2rs:ephemeral true) are present in the dynamic datastore.

5.4. Define how dynamic data is actualized

The diagram in Section 4 depicts dynamic datastores feeding into the <operational> datastore. How this interaction occurs must be defined by the dynamic datastore. In some cases, it may occur implicitly, as soon as the data is put into the dynamic datastore while, in other cases, an explicit action (e.g., an RPC) may be required to trigger the application of the dynamic datastore’s data.

5.5. Define which protocols can be used

By default, it is assumed that both the NETCONF and RESTCONF protocols can be used to interact with a dynamic datastore. However, it may be that only a specific protocol can be used (e.g., Forces) or that a subset of all protocol operations or capabilities are available (e.g., no locking, no xpath-based filtering, etc.).

5.6. Define a module for the dynamic datastore

Each dynamic datastore MUST be defined by a YANG module. This module is used by servers to indicate (e.g., via YANG Library) their support for the dynamic datastore.

The YANG module MUST import the "ietf-datastores" and "ietf-origin" modules, defined in this document. This is necessary in order to access the base identities they define.

The YANG module MUST define an identity that uses the "ds:datastore" identity as its base. This identity is necessary so that the datastore can be referenced in protocol operations (e.g., <get-data>).

The YANG module MUST define an identity that uses the "or:dynamic" identity as its base. This identity is necessary so that data

originating from the datastore can be identified as such via the "origin" metadata attribute defined in Section 6.

An example of these guidelines in use is provided in Appendix B.

6. YANG Modules

<CODE BEGINS> file "ietf-datastores@2017-03-13.yang"

```
module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

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

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

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

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

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

     Author:   Phil Shafer
               <mailto:phil@juniper.net>

     Author:   Kent Watsen
               <mailto:kwatsen@juniper.net>

     Author:   Rob Wilton
               <rwilton@cisco.com>";

  description
    "This YANG module defines a set of identities for datastores.
     These identities can be used to identify datastores in protocol
     operations.

     Copyright (c) 2017 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
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

    revision 2017-03-13 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: Network Management Datastore Architecture";
    }

    /*
     * Identities
     */

    identity datastore {
      description
       "Abstract base identity for datastore identities.";
    }

    identity static {
      description
       "Abstract base identity for static configuration datastores.";
    }

    identity dynamic {
      description
       "Abstract base identity for dynamic configuration datastores.";
    }

    identity running {
      base static;
      description
       "The 'running' datastore.";
    }

    identity candidate {
      base static;
      description
       "The 'candidate' datastore.";
    }

    identity startup {
      base static;
```

```
      description
       "The 'startup' datastore.";
    }

    identity intended {
      base static;
      description
       "The 'intended' datastore.";
    }

    identity operational {
      base datastore;
      description
       "The 'operational' state datastore.";
    }

  }

  <CODE ENDS>

  <CODE BEGINS> file "ietf-datastores@2017-03-13.yang"

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

    import ietf-yang-metadata {
      prefix md;
    }

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

    contact
      "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

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

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

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

       Author:   Phil Shafer
                 <mailto:phil@juniper.net>
```

```
        Author:   Kent Watsen
                  <mailto:kwatsen@juniper.net>

        Author:   Rob Wilton
                  <rwilton@cisco.com>";

     description
       "This YANG module defines an 'origin' metadata annotation, and a
        set of identities for the origin value.  The 'origin' metadata
        annotation is used to mark data in the 'operational'
        datastore with information on where the data originated.

        Copyright (c) 2017 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
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

     revision 2017-03-13 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: Network Management Datastore Architecture";
     }

     /*
      * Identities
      */

     identity origin {
       description
         "Abstract base identity for the origin annotation.";
     }

     identity static {
       base origin;
       description
         "Denotes data from static configuration (e.g., <intended>).";
     }
```

```
     identity dynamic {
       base origin;
       description
         "Denotes data from dynamic configuration protocols
          or dynamic datastores (e.g., DHCP).";
     }

     identity system {
       base origin;
       description
         "Denotes data created by the system independently of what
          has been configured.";
     }

     identity default {
       base origin;
       description
         "Denotes data that does not have an explicitly configured
          value, but has a default value in use.  Covers both simple
          defaults and defaults defined via an explanation in a
          description statement.";
     }

     /*
      * Metadata annotations
      */

     md:annotation origin {
       type identityref {
         base origin;
       }
     }

   }
```

<CODE ENDS>

7. IANA Considerations

7.1. Updates to the IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-datastores
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-origin
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
```

7.2. Updates to the YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registrations are requested:

```
name:         ietf-datastores
namespace:    urn:ietf:params:xml:ns:yang:ietf-datastores
prefix:       ds
reference:    RFC XXXX

name:         ietf-origin
namespace:    urn:ietf:params:xml:ns:yang:ietf-origin
prefix:       or
reference:    RFC XXXX
```

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.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:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Andy Bierman, YumaWorks, <andy@yumaworks.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, Google, <robjs@google.com>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

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>.

[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>.

[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>.

[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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <http://www.rfc-editor.org/info/rfc7952>.

[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>.

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.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-12 (work in progress), March 2017.

[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.

[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>.

[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. Example Data

The use of datastores is complex, and many of the subtle effects are more easily presented using examples. This section presents a series of example data models with some sample contents of the various datastores.

A.1. System Example

In this example, the following fictional module is used:

```
module example-system {
  yang-version 1.1;
  namespace urn:example:system;
  prefix sys;

  import ietf-inet-types {
    prefix inet;
  }

  container system {
    leaf hostname {
      type string;
    }

    list interface {
      key name;

      leaf name {
        type string;
      }

      container auto-negotiation {
        leaf enabled {
          type boolean;
          default true;
        }
        leaf speed {
          type uint32;
          units mbps;
          description
            "The advertised speed, in mbps.";
        }
      }
```

```
        leaf speed {
          type uint32;
          units mbps;
          config false;
          description
            "The speed of the interface, in mbps.";
        }

        list address {
          key ip;

          leaf ip {
            type inet:ip-address;
          }
          leaf prefix-length {
            type uint8;
          }
        }
      }
    }
  }
```

The operator has configured the host name and two interfaces, so the contents of <intended> is:

```
   <system xmlns="urn:example:system">

     <hostname>foo</hostname>

     <interface>
       <name>eth0</name>
       <auto-negotiation>
         <speed>1000</speed>
       </auto-negotiation>
       <address>
         <ip>2001:db8::10</ip>
         <prefix-length>32</prefix-length>
       </address>
     </interface>

     <interface>
       <name>eth1</name>
       <address>
         <ip>2001:db8::20</ip>
         <prefix-length>32</prefix-length>
       </address>
     </interface>

   </system>
```

The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. In addition to a default value, a loopback interface is automatically added by the system, and the result of the "speed" auto-negotiation. All of this is reflected in <operational>:

```
   <system
       xmlns="urn:example:system"
       xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

     <hostname or:origin="or:dynamic">bar</hostname>

     <interface or:origin="or:static">
       <name>eth0</name>
       <auto-negotiation>
         <enabled or:origin="or:default">true</enabled>
         <speed>1000</speed>
       </auto-negotiation>
       <speed>100</speed>
       <address>
         <ip>2001:db8::10</ip>
         <prefix-length>32</prefix-length>
       </address>
       <address or:origin="or:dynamic">
         <ip>2001:db8::1:100</ip>
         <prefix-length>32</prefix-length>
       </address>
     </interface>

     <interface or:origin="or:system">
       <name>lo0</name>
       <address>
         <ip>::1</ip>
         <prefix-length>128</prefix-length>
       </address>
     </interface>

   </system>
```

A.2. BGP Example

Consider the following piece of a ersatz BGP module:

```
      container bgp {
        leaf local-as {
          type uint32;
        }
        leaf peer-as {
          type uint32;
        }
        list peer {
          key name;
          leaf name {
            type ipaddress;
          }
          leaf local-as {
            type uint32;
            description
              ".... Defaults to ../local-as";
          }
          leaf peer-as {
            type uint32;
            description
               "... Defaults to ../peer-as";
          }
          leaf local-port {
            type inet:port;
          }
          leaf remote-port {
            type inet:port;
            default 179;
          }
          leaf state {
            config false;
            type enumeration {
              enum init;
              enum established;
              enum closing;
            }
          }
        }
      }
```

In this example model, both bgp/peer/local-as and bgp/peer/peer-as have complex hierarchical values, allowing the user to specify default values for all peers in a single location.

The model also follows the pattern of fully integrating state ("config false") nodes with configuration ("config true") nodes. There is not separate "bgp-state" hierarchy, with the accompanying

repetition of containment and naming nodes. This makes the model simpler and more readable.

A.2.1. Datastores

Each datastore represents differing views of these data nodes. The <running> datastore will hold the configuration data provided by the user, for example a single BGP peer. The <intended> datastore will conceptually hold the data as validated, after the removal of data not intended for validation and after any local template mechanisms are performed. The <operational> datastore will show data from <intended> as well as any "config false" nodes.

A.2.2. Adding a Peer

If the user configures a single BGP peer, then that peer will be visible in both the <running> and <intended> datastores. It may also appear in the <candidate> datastore, if the server supports the "candidate" feature. Retrieving the peer will return only the user-specified values.

No time delay should exist between the appearance of the peer in <running> and <intended>.

In this scenario, we've added the following to <running>:

```
<bgp>
  <local-as>64642</local-as>
  <peer-as>65000</peer-as>
  <peer>
    <name>10.1.2.3</name>
  </peer>
</bgp>
```

A.2.2.1. <operational>

The <operational> datastore will contain the fully expanded peer data, including "config false" nodes. In our example, this means the "state" node will appear.

In addition, the <operational> datastore will contain the "currently in use" values for all nodes. This means that local-as and peer-as will be populated even if they are not given values in <intended>. The value of bgp/local-as will be used if bgp/peer/local-as is not provided; bgp/peer-as and bgp/peer/peer-as will have the same relationship. In the operational view, this means that every peer will have values for their local-as and peer-as, even if those values

are not explicitly configured but are provided by bgp/local-as and bgp/peer-as.

Each BGP peer has a TCP connection associated with it, using the values of local-port and remote-port from the intended datastore. If those values are not supplied, the system will select values. When the connection is established, the <operational> datastore will contain the current values for the local-port and remote-port nodes regardless of the origin. If the system has chosen the values, the "origin" attribute will be set to "operational". Before the connection is established, one or both of the nodes may not appear, since the system may not yet have their values.

```
<bgp origin="or:static" xmlns="urn:example:bgp">
  <local-as origin="or:static">64642</local-as>
  <peer-as origin="or:static">65000</peer-as>
  <peer origin="or:static">
    <name origin="or:static">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:system">60794</local-port>
    <remote-port origin="or:default">179</remote-port>
  </peer>
</bgp>
```

A.2.3. Removing a Peer

Changes to configuration data may take time to percolate through the various software components involved. During this period, it is imperative to continue to give an accurate view of the working of the device. The <operational> datastore will return data nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device.

Consider the scenario where a client removes a BGP peer. When a peer is removed, the operational state will continue to reflect the existence of that peer until the peer’s resources are released, including closing the peer’s connection. During this period, the current data values will continue to be visible in the <operational> datastore, with the "origin" attribute set to indicate the origin of the original data.

```
<bgp origin="or:static">
  <local-as origin="or:static">64642</local-as>
  <peer-as origin="or:static">65000</peer-as>
  <peer origin="or:static">
    <name origin="or:static">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:static">60794</local-port>
    <remote-port origin="or:static">179</remote-port>
  </peer>
</bgp>
```

Once resources are released and the connection is closed, the peer's data is removed from the <operational> datastore.

A.3. Interface Example

In this section, we'll use this simple interface data model:

```
container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint;
    }
    leaf ipv4-address {
      type inet:ipv4-address;
    }
  }
}
```

A.3.1. Pre-provisioned Interfaces

One common issue in networking devices is the support of Field Replaceable Units (FRUs) that can be inserted and removed from the device without requiring a reboot or interfering with normal operation. These FRUs are typically interface cards, and the devices support pre-provisioning of these interfaces.

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, then the <intended> datastore might contain the following:

```
  <interfaces>
    <interface>
      <name>et-0/0/0</name>
      <description>Test interface</description>
    </interface>
  </interfaces>
```

Since the interface does not exist, this data does not appear in the <operational> datastore.

When a FRU containing this interface is inserted, the system will detect it and process the associated configuration. The <operational> will contain the data from <intended>, as well as the "config false" nodes, such as the current value of the interface's MTU.

```
  <interfaces origin="or:static">
    <interface origin="or:static">
      <name origin="or:static">et-0/0/0</name>
      <description origin="or:static">Test interface</description>
      <mtu origin="or:system">1500</mtu>
    </interface>
  </interfaces>
```

If the FRU is removed, the interface data is removed from the <operational> datastore.

A.3.2. System-provided Interface

Imagine if the system provides a loopback interface (named "lo0") with a default ipv4-address of "127.0.0.1". The system will only provide configuration for this interface if the is no data for it in <intended>.

When no configuration for "lo0" appears in <intended>, then <operational> will show the system-provided data:

```
  <interfaces origin="or:static">
    <interface origin="or:system">
      <name origin="or:system">lo0</name>
      <ipv4-address origin="or:system">127.0.0.1</ipv4-address>
    </interface>
  </interfaces>
```

When configuration for "lo0" does appear in <intended>, then <operational> will show that data with the origin set to "intended". If the "ipv4-address" is not provided, then the system-provided value will appear as follows:

```
<interfaces origin="or:static">
  <interface origin="or:static">
    <name origin="or:static">lo0</name>
    <description origin="or:static">loopback</description>
    <ipv4-address origin="or:system">127.0.0.1</ipv4-address>
  </interface>
</interfaces>
```

Appendix B. Ephemeral Dynamic Datastore Example

The section defines documentation for an example dynamic datastore using the guidelines provided in Section 5. While this example is very terse, it is expected to be that a standalone RFC would be needed when fully expanded.

This example defines a dynamic datastore called "ephemeral", which is loosely modeled after the work done in the I2RS working group.

1. Name : ephemeral
2. YANG modules : all (default)
3. YANG statements : config false + ephemeral true
4. How applied : automatic
5. Protocols : NC/RC (default)
6. YANG Module : (see below)

```
   module example-ds-ephemeral {
     yang-version 1.1;
     namespace "urn:example:ds-ephemeral";
     prefix eph;

     import ietf-datastores {
       prefix ds;
     }
     import ietf-origin {
       prefix or;
     }

     // add datastore identity
     identity ds-ephemeral {
       base ds:datastore;
       description
         "The 'ephemeral' datastore.";
     }

     // add origin identity
     identity or-ephemeral {
       base or:dynamic;
       description
         "Denotes data from the ephemeral dynamic datastore.";
     }

     // define ephemeral extension
     extension ephemeral {
       argument "value";
       description
         "This extension is mixed into config false YANG nodes to
          indicate that they are writable nodes in the 'ephemeral'
          datastore.  This statement takes a single argument
          representing a boolean having the values 'true' and 'false'.
          The default value is 'false'.";
     }
   }
```

Appendix C. 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 datastore model described here this is not needed anymore since the different datastores handle the different lifetimes of data objects. Introducing this model together with the

deprecation of the <get/> operation makes it possible to write simpler models.

C.1. Proposed migration of existing YANG Data Models

For standards based YANG modules that have already been published, that are using split config and state trees, it is planned that these modules are updated with new revisions containing the following changes:

- The top level module description is updated to indicate that the module conforms to the revised datastore architecture with a combined config and state tree, and that the existing state tree nodes are deprecated, to be obsoleted over time.

- All status "current" data nodes under the existing "state" trees are copied to the equivalent place under the "config" tree:

  * If a node with the same name and type already exists under the equivalent path in the config tree then the nodes are merged and the description updated.

  * If a node with the same name but different type exists under the equivalent path in the config tree, then the module authors must choose the appropriate mechanism to combine the config and state nodes in a backwards compatible way based on the data model design guidelines below. This may require the state node to be added to the config tree with a modified name. This scenario is expected to be relatively uncommon.

  * If no node with the same name and path already exists under the config tree then the state node schema is copied verbatim into the config tree.

  * As the state nodes are copied into the config trees, any leafrefs that reference other nodes in the state tree are adjusted to reference the equivalent path in the config tree.

  * All status "current" nodes under the existing "state" trees are marked as "status" deprecated.

- Augmentations are similarly handled to data nodes as described above.

C.2. Standardization of new YANG Data Models

New standards based YANG modules, or those in active development, should be designed to conform to the revised datastore architecture, following the design guidelines described below, and only need to provide combined config/state trees.

Appendix D. Implications on other Documents

The sections below describe the authors' thoughts on how various other documents may be updated to support the datastore architecture described in this document. They have been incorporated as an appendix of this document to facilitate easier review, but the expectation is that this work will be moved into another document as soon as the appropriate working group decides to take on the work.

D.1. Implications on YANG

Note: This section describes the authors' thoughts on how YANG [RFC7950] could be updated to support the datastore architecture described in this document. It has been incorporated here as a temporary measure to facilitate easier review, but the expectation is that this work will be owned and standardized via the NETCONF working group.

- Some clarifications may be needed if this datastore model is adopted. YANG currently describes validation in terms of the <running> configuration datastore while it really happens on the <intended> configuration datastore.

D.2. Implications on YANG Library

Note: This section describes the authors' thoughts on how YANG Library [RFC7895] could be updated to support the datastore architecture described in this document. It has been incorporated here as a temporary measure to facilitate easier review, but the expectation is that this work will be owned and standardized via the NETCONF working group.

With the introduction of multiple datastores, it is important that a server can advertise to clients which modules are supported in the different datastores implemented by the server. In order to do this, we propose that the "ietf-yang-module" ([RFC7895]) is revised, with the following addition to the "module" list in the "module-list" grouping:

```
    leaf-list datastore {
      type identityref {
        base ds:datastore;
      }
      description
        "The datastores in which this module is supported.";

    }
```

D.3. Implications to YANG Guidelines

Note: This section describes the authors' thoughts on how Guidelines for Authors and Reviewers of YANG Data Model Documents [I-D.ietf-netmod-rfc6087bis] could be updated to support the datastore architecture described in this document. It has been incorporated here as a temporary measure to facilitate easier review, but the expectation is that this work will be owned and standardized via the NETCONF working group.

It is important to design data models with clear semantics that work equally well for instantiation in a configuration datastore and instantiation in the <operational> datastore.

D.3.1. Nodes with different config/state value sets

There may be some differences in the value set of some nodes 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 different nodes for the configured and state values.

D.3.2. 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, it is recommended to model this as two separate leafs, one config true leaf for the input to the auto-negotiation process, and one config false leaf for the output from the process.

D.4. Implications on NETCONF

Note: This section describes the authors' thoughts on how NETCONF [RFC6241] could be updated to support the datastore architecture described in this document. It has been incorporated here as a temporary measure to facilitate easier review, but the expectation is

that this work will be owned and standardized via the NETCONF working group.

D.4.1. Introduction

The NETCONF protocol [RFC6241] defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated.

NETCONF already has support for configuration datastores, but it does not define an operational datastore. Instead, it provides the <get> operation that returns the contents of the <running> datastore along with all config false leaves. However, this <get> operation is incompatible with the new datastore architecture defined in this document, and hence should be deprecated.

There are two possible ways that NETCONF could be extended to support the new architecture: Either as new optional capabilities extending the current version of NETCONF (v1.1, [RFC6241]), or by defining a new version of NETCONF.

Many of the required additions are common to both approaches, and are described below. A following section then describes the benefits of defining a new NETCONF version, and the additional changes that would entail.

D.4.2. Overview of additions to NETCONF

- A new "supported datastores" capability allows a device to list all datastores it supports. Implementations can choose which datastores they expose, but MUST at least expose both the <running> and <operational> datastores. They MAY expose additional datastores, such as <intended>, <candidate>, etc.

- A new <get-data> operation is introduced that allows the client to return the contents of a datastore. For configuration datastores, this operation returns the same data that would be returned by the existing <get-config> operation.

- Some form of new filtering mechanism is required to allow the device to filter the data based on the YANG metadata in addition to other filters (such as the subtree filter). See also Appendix E.

- A new "with-metadata" capability allows a device to indicate that it supports the capability of including YANG metadata annotations in the responses to <get> and <get-config> requests. This is

achieved in a similar way to with-defaults [RFC6243], by introducing a <with-metadata> XML element to <get> and <get-config> requests.

* The capability would allow a device to indicate which types of metadata are supported.

* The XML element would specify which types of metadata are included in the response.

o The handling of defaults for the new configuration datastores is as described in with-defaults [RFC6243], but that does not apply for the operational state datastore that defines new semantics.

D.4.2.1. Operational State Datastore Defaults Handling

The normal semantics for the <operational> datastore are that all values that match the default specified in the schema are included in response to requests on the operational state datastore. This is equivalent to the "report-all" mode of the with-defaults handling.

The "metadata-filter" query parameter can be used to exclude nodes with origin metadata matching "default", that would exclude nodes that match the default value specified in the schema.

If the server cannot return a value for any reason (e.g., the server cannot determine the value, or the value that would be returned is outside the allowed leaf value range) then the server can choose to not return any value for a particular leaf, which MUST be interpreted by the client as the value of that leaf not being known, rather than implicitly having the default value.

D.4.3. Overview of NETCONF version 2

This section describes NETCONF version 2, by explaining the differences to NETCONF version 1.1. Where not explicitly specified, the behavior of NETCONF version 2 is the same as for NETCONF version 1.1 [RFC6241].

D.4.3.1. Benefits of defining a new NETCONF version

Defining a new version of NETCONF (as opposed to extending NETCONF version 1.1) has several benefits:

o It allows for removal of the existing <get> RPC operation, that returns content from both the running configuration datastore combined with all config false leaves.

- It could allow the existing <get-config> operation to also be removed, replaced by the more generic <get-data> that is named appropriately to also apply to the operational datastore.

- It makes it easier for clients and servers to know what reasonable common baseline functionality to expect, rather than a collection of capabilities that may not be implemented in a consistent fashion. In particular, clients will able to assume support for the <operational> datastore.

- It can gracefully coexist with NETCONF v1.1. A server could implement both versions. Existing YANG models exposing split config/state trees could be exposed via NETCONF v1.1, whereas combined config/state YANG models could be exposed via NETCONF v2, providing a viable server upgrade path.

D.4.3.2. Proposed changes for NETCONF v2

The differences between NETCONF v2 and NETCONF v1.1 can be summarized as:

- NETCONF v2 advertises a new base NETCONF capability "urn:ietf:params:netconf:base:2.0". A server may advertise older NETCONF versions as well, to allow a client to choose which version to use.

- NETCONF v2 removes support for the existing <get> operation, that is replaced by the <get-data> on the operational datastore.

- NETCONF v2 can publish a separate version of YANG library from a NETCONF v1.1 implementation running on the same device, allowing different versions of NETCONF to support a different set of YANG modules.

D.4.3.3. Possible Migration Paths

A common approach in current data models is to have two separate trees "/foo" and "/foo-state", where the former contains config true nodes, and the latter config false nodes. A data model that is designed for the revised architectural framework presented in this document will have a single tree "/foo" with a combination of config true and config false nodes.

Two different migration strategies are considered:

D.4.3.3.1. Migration Path using two instances of NETCONF

If, for backwards compatability reasons, a server intends to support both split config/state trees and the combined config/state trees proposed in this architecture, then this can be achieved by having the device support both NETCONF v1 and NETCONF v2 at the same time:

- The NETCONF v1 implementation could support existing YANG module revisions defined with split config/state trees.
- The NETCONF v2 implementation could support different YANG modules, or YANG module revisions, with combined config/state trees.

Clients can then decide on which type of models to use by expressing the appropriate version of the base NETCONF capability during capability exchange.

D.4.3.3.2. Migration Path using a single instance of NETCONF

The proposed strategy for updating existing published data models is to publish new revisions with the state trees' nodes copied under the config tree, and for the existing state trees to have all of their nodes marked as deprecated. The expectation is that NETCONF servers would use a combination of these updated models alongside new models that only follow the new datastore architecture.

- NETCONF servers can support clients that are not aware of the revised datastore architecture, particularly if they continue to support the deprecated <get> operation:
  - For updated YANG modules they would see additional information returned via the <get> operation.
  - For new YANG modules, some of the state nodes may not be available, i.e. for any state nodes that exist under a config node that has not been configured (e.g., statistics under a system created interface).
- NETCONF servers can also support clients that are aware of the revised datastores architecture:
  - For updated YANG modules they would see additional information returned under the legacy state trees. This information can be excluded using appropriate subtree filters.
  - New YANG modules, conforming to the datastores architecture, would work exactly as expected.

D.5. Implications on RESTCONF

This section describes the authors' thoughts on how RESTCONF [RFC8040] could be updated to support the datastore architecture described in this document. It has been incorporated here as a temporary measure to facilitate easier review, but the expectation is that this work will be owned and standardized via the NETCONF working group.

D.5.1. Introduction

RESTCONF [RFC8040] defines a protocol based on HTTP for configuring data defined in YANG version 1 or 1.1, using a conceptual datastore that is compatible with a server that implements NETCONF 1.1 compliant datastores.

The combined conceptual datastore defined in RESTCONF is incompatible with the new datastore architecture defined in this document. There are two possible ways that RESTCONF could be extended to support the new architecture: Either as new optional capabilities extending the existing RESTCONF RFC, or possibly as an new version of RESTCONF.

Many of the required additions are common to both approaches, and are described below. A following section then describes the potential benefits of defining a new RESTCONF version, and the additional changes that might entail.

D.5.2. Overview of additions to RESTCONF

- A new path {+restconf}/datastore/<datastore-name>/data/ to provide a YANG data tree for each datastore that is exposed via RESTCONF.

- Implementations can choose which datastores they expose, but MUST at least expose both the <running> and <operational> datastores. They MAY expose the <intended> datastores as needed.

- The same HTTP Methods supported on {+restconf}/data/ are also supported on {+restconf}/datastore/<datastore-name>/data/ but suitably constrained depending on whether the datastore can be written to by the client, or is read-only.

- The same query parameters supported on {+restconf}/data/ are also support on {+restconf}/datastore/<datastore-name>/data/ except for the following query parameters:

- "metadata" - is a new optional query parameter that filters the returned data based on the metadata annotation.

- "with-metadata" - is a new optional query parameter that indicating that the metadata annotations should be included in the reply.

- "with-defaults" is supported on all configuration datastores, but is not supported on the operational state datastore path, because it has different default handling semantics.

- The handling of defaults (include the with-defaults query parameter) for the new configuration datastores is the same as the existing conceptual datastore, but does not apply for the operational state datastore that defines new semantics.

D.5.2.1. HTTP Methods

All configuration datastores support all HTTP Methods.

The <operational> datastore only supports the following HTTP methods: OPTIONS, HEAD, GET, and POST to invoke an RFC operation.

D.5.2.2. Query parameters

[RFC7952] specifies how a YANG data tree can be annotated with generic metadata information, that is used by this document to annotate data nodes with origin information indicating the mechanism by which the operational value came into effect.

RESTCONF could be extended with an optional generic mechanism to allow the filtering of nodes returned in a query based on metadata annotations associated with the data node.

RESTCONF could also be extended with an optional generic mechanism to choose whether metadata annotations should be included in the response, potentially filtering to a subset of annotations. E.g., only include @origin metadata annotations, and not any others that may be in use.

Both of the generic mechanisms could be controlled by a new capability. A new capability is defined to indicate whether a device supports filtering on, or annotating responses with, the origin meta data.

D.5.2.3. Operational State Datastore Defaults Handling

The normal semantics for the <operational> datastore are that all values that match the default specified in the schema are included in response to requests on the operational state datastore. This is equivalent to the "report-all" mode of the with-defaults handling.

The "metadata" query parameter can be used to exclude nodes with a origin metadata matching "default", that would exclude (only config true?) nodes that match the default value specified in the schema.

If the server cannot return a value for any reason (e.g., the server cannot determine the value, or the value that would be returned is outside the allowed leaf value range) then the server can choose to not return any value for a particular leaf, which MUST be interpreted by the client as the value of that leaf not being known, rather than implicitly having the default value.

D.5.3. Overview of a possible new RESTCONF version

This section describes a notional new RESTCONF version, by explaining the differences to RESTCONF version 1. Where not explicitly specified, the behavior of a new RESTCONF version is the same as for RESTCONF version 1 [RFC8040].

D.5.3.1. Potential benefits of defining a new RESTCONF version

Defining a new version of RESTCONF (as opposed to extending RESTCONF version 1) has several potential benefits:

- It could expose datastores, and models designed for the revised datastore architecture, in a clean and consistent way.
- It would allow the parts of RESTCONF that do not work well with the revised datastore architecture to be omitted from the new RESTCONF version.
- It would make it easier for clients and servers to know what reasonable common baseline functionality to expect, rather than a collection of capabilities that may not be implemented in a consistent fashion.
- It could gracefully coexist with RESTCONF v1. A server could implement both versions. Existing YANG models exposing split config/state trees could be exposed via RESTCONF v1, whereas combined config/state YANG models could be exposed via a new RESTCONF version, providing a viable server upgrade path.

D.5.3.2. Possible changes for a new RESTCONF version

The differences between a notional new RESTCONF version and RESTCONF version 1 (RESTCONF v1) [RFC8040] can be summarized as:

- A new RESTCONF version would define a new root resource, and a separate link relation in the /.well-known/host-meta resource.

- o A new RESTCONF version could remove support for the {+restconf}/data path supported in RESTCONF v1.

- o A new RESTCONF version could publish a separate version of YANG library from a RESTCONF v1 implementation running on the same device, allowing different versions of RESTCONF to support a different set of YANG modules.

D.5.3.3. Possible Migration Path using a new RESTCONF version

A common approach in current data models is to have two separate trees "/foo" and "/foo-state", where the former contains config true nodes, and the latter config false nodes. A data model that is designed for the revised architectural framework presented in this document will have a single tree "/foo" with a combination of config true and config false nodes.

If for backwards compatability reasons, a server intends to support both split config/state trees, and the combined config/state trees proposed in this architecture, then this could be achieved by having the device support both RESTCONF v1 and the new RESTCONF version at the same time:

- o The RESTCONF v1 implementation could support existing YANG module revisions defined with split config/state trees.

- o The implementation of the new RESTCONF version could support different YANG modules, or YANG module revisions, with combined config/state trees.

Clients can then decide on which type of models to use by choosing whether to use the RESTCONF v1 root resource or the root resource associated with the new RESTCONF version.

Appendix E. Open Issues

1. NETCONF needs to be able to filter data based on the origin metadata. Possibly this could be done as part of the <get-data> operation.

2. We need a means of inheriting @origin values, so whole hierarchies can avoid the noise of repeating parent values. Should "origin=’system’" (or whatever we call it) be the default?

3. We need to discuss somewhere how remote procedure calls and notifications/actions tie into datastores. RFC 7950 shows as an example a ping action tied to an interface. Does this refer to an interface defined in a configuration datastore? Or an

interface defined in the operational state datastore? Or the applied configuration datastore? Similarly, RFC 7950 shows an example of a link-failure notification; this likely applies implicitly to the operational state datastore. The netconf-config-change notification does explicitly identify a datastore. I think we generally need to have remote procedure calls and notifications be explicit about which datastores they apply to and perhaps change the default xpath context from running plus state to the operational state datastore.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Rob Wilton
Cisco Systems

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: July 17, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
January 13, 2018

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-10

Abstract

Datastores are a fundamental concept binding the data models written in the YANG data modeling language to network management protocols such as NETCONF and RESTCONF. This document defines an architectural framework for datastores based on the experience gained with the initial simpler model, addressing requirements that were not well supported in the initial model. This document updates RFC 7950.

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 July 17, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction 3
2. Objectives 3
3. Terminology 4
4. Background 7
4.1. Original Model of Datastores 8
5. Architectural Model of Datastores 9
5.1. Conventional Configuration Datastores 10
5.1.1. The Startup Configuration Datastore (<startup>) . . . 11
5.1.2. The Candidate Configuration Datastore (<candidate>) . 11
5.1.3. The Running Configuration Datastore (<running>) . . . 12
5.1.4. The Intended Configuration Datastore (<intended>) . . 12
5.2. Dynamic Configuration Datastores 13
5.3. The Operational State Datastore (<operational>) 13
5.3.1. Remnant Configuration 14
5.3.2. Missing Resources 15
5.3.3. System-controlled Resources 15
5.3.4. Origin Metadata Annotation 15
6. Implications on YANG 17
6.1. XPath Context 17
6.2. Invocation of Actions and RPCs 18
7. YANG Modules 18
8. IANA Considerations 24
8.1. Updates to the IETF XML Registry 24
8.2. Updates to the YANG Module Names Registry 24
9. Security Considerations 24
10. Acknowledgments 25
11. References 25
11.1. Normative References 26
11.2. Informative References 26
Appendix A. Guidelines for Defining Datastores 27
A.1. Define which YANG modules can be used in the datastore . 27
A.2. Define which subset of YANG-modeled data applies 28
A.3. Define how data is actualized 28
A.4. Define which protocols can be used 28
A.5. Define YANG identities for the datastore 28
Appendix B. Ephemeral Dynamic Configuration Datastore Example . 29
Appendix C. Example Data 30
C.1. System Example 30

C.2. BGP Example 33
C.2.1. Datastores 35
C.2.2. Adding a Peer 35
C.2.3. Removing a Peer 36
C.3. Interface Example 37
C.3.1. Pre-provisioned Interfaces 37
C.3.2. System-provided Interface 38
Authors' Addresses 39

1. Introduction

This document provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols. 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. This architecture is agnostic with regard to the encoding used by network management protocols.

This document updates RFC 7950 by refining the definition of the accessible tree for some XPath context (see Section 6.1) and the invocation context of operations (see Section 6.2).

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

2. Objectives

Network management data objects can often take two different values, the value configured by the user or an application (configuration) and the value that the device is actually using (operational state). These two values may be different for a number of reasons, e.g., system internal interactions with hardware, interaction with protocols or other devices, or simply the time it takes to propagate a configuration change to the software and hardware components of a system. Furthermore, configuration and operational state data objects may have different lifetimes.

The original model of datastores required these data objects to be modeled twice in the YANG schema, as "config true" objects and as "config false" objects. The convention adopted by the interfaces

data model ([RFC7223]) and the IP data model ([RFC7277]) was using two separate branches rooted at the root of the data tree, one branch for configuration data objects and one branch for operational state data objects.

The duplication of definitions and the ad-hoc separation of operational state data from configuration data leads to a number of problems. Having configuration and operational state data in separate branches in the data model is operationally complicated and impacts the readability of module definitions. Furthermore, the relationship between the branches is not machine readable and filter expressions operating on configuration and on related operational state are different.

With the revised architectural model of datastores defined in this document, the data objects are defined only once in the YANG schema but independent instantiations can appear in different datastores, e.g., one for a configured value and another for an operationally used value. This provides a more elegant and simpler solution to the problem.

The revised architectural model of datastores supports additional datastores for systems that support more advanced processing chains converting configuration to operational state. For example, some systems support configuration that is not currently used (so called inactive configuration) or they support configuration templates that are used to expand configuration data via a common template.

3. Terminology

This document defines the following terminology. Some of the terms are revised definitions of terms originally defined in [RFC6241] and [RFC7950] (see also section Section 4). The revised definitions are semantically equivalent with the definitions found in [RFC6241] and [RFC7950]. It is expected that the revised definitions provided in this section will replace the definitions in [RFC6241] and [RFC7950] when these documents are revised.

- 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. A datastore maps to an instantiated YANG data tree.

- schema node: A node in the schema tree. The formal definition is in RFC 7950.

- datastore schema: The combined set of schema nodes for all modules supported by a particular datastore, taking into consideration any deviations and enabled features for that datastore.

- configuration: Data that is required to get a device from its initial default state into a desired operational state. This data is modeled in YANG using "config true" nodes. Configuration can originate from different sources.

- configuration datastore: A datastore holding configuration.

- running configuration datastore: A configuration datastore holding the current configuration of the device. It may include configuration that requires further transformations before it can be applied. This datastore is referred to as "<running>".

- candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's running configuration datastore and that can be committed to the running configuration datastore. This datastore is referred to as "<candidate>".

- startup configuration datastore: A configuration datastore holding the configuration loaded by the device into the running configuration datastore when it boots. This datastore is referred to as "<startup>".

- intended configuration: Configuration that is intended to be used by the device. It represents the configuration after all configuration transformations to <running> have been performed and is the configuration that the system attempts to apply.

- intended configuration datastore: A configuration datastore holding the complete intended configuration of the device. This datastore is referred to as "<intended>".

- configuration transformation: The addition, modification or removal of configuration between the <running> and <intended> datastores. Examples of configuration transformations include the removal of inactive configuration and the configuration produced through the expansion of templates.

- conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, and <intended>. These datastores share a common datastore schema, and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.

- conventional configuration: Configuration that is stored in any of the conventional configuration datastores.

- dynamic configuration datastore: A configuration datastore holding configuration obtained dynamically during the operation of a device through interaction with other systems, rather than through one of the conventional configuration datastores.

- dynamic configuration: Configuration obtained via a dynamic configuration datastore.

- learned configuration: Configuration that has been learned via protocol interactions with other systems and that is neither conventional nor dynamic configuration.

- system configuration: Configuration that is supplied by the device itself.

- default configuration: Configuration that is not explicitly provided but for which a value defined in the data model is used.

- applied configuration: Configuration that is actively in use by a device. Applied configuration originates from conventional, dynamic, learned, system and default configuration.

- system state: The additional data on a system that is not configuration, such as read-only status information and collected statistics. System state is transient and modified by interactions with internal components or other systems. System state is modeled in YANG using "config false" nodes.

- operational state: The combination of applied configuration and system state.

- operational state datastore: A datastore holding the complete operational state of the device. This datastore is referred to as "<operational>".

- origin: A metadata annotation indicating the origin of a data item.

- remnant configuration: Configuration that remains part of the applied configuration for a period of time after it has been removed from the intended configuration or dynamic configuration. The time period may be minimal, or may last until all resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles) have been deallocated.

The following additional terms are not datastore specific but commonly used and thus defined here as well:

- client: An entity that can access YANG-defined data on a server, over some network management protocol.
- server: An entity that provides access to YANG-defined data to a client, over some network management protocol.
- notification: A server-initiated message indicating that a certain event has been recognized by the server.
- remote procedure call: An operation that can be invoked by a client on a server.

4. Background

NETCONF [RFC6241] provides the following definitions:

- 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.
- configuration datastore: The datastore holding the complete set of configuration that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [RFC7950] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG existed):

- datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.
- configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration.

[RFC6244] defined operational state data as follows:

- 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 [RFC6244] discusses operational state and among other things mentions the option to consider operational state as

being stored in another datastore. Section 4.4 of [RFC6244] then concludes that at the time of the writing, modeling state as distinct leafs and distinct branches is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] 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.

4.1. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:

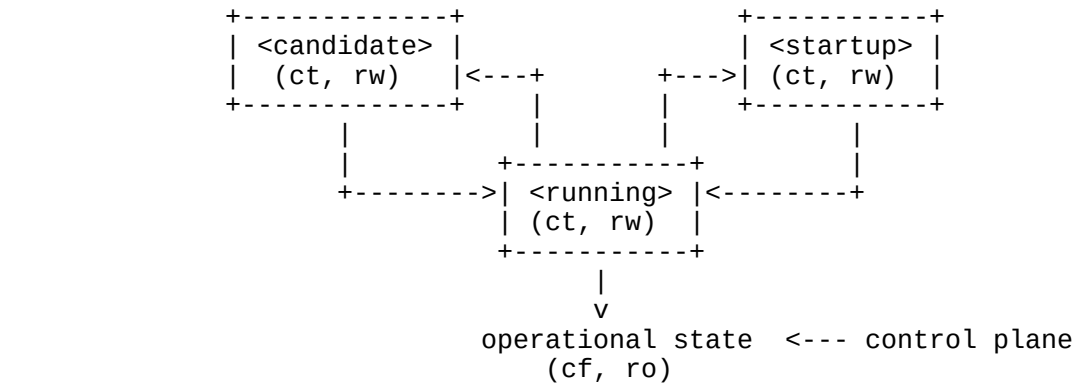


Figure 1

Note that this diagram simplifies the model: read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for <candidate> and <startup> is optional and <running> does not have to be writable. Furthermore, <startup> 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 <candidate> or by directly modifying <running> 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 <running>. 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 contents of <running> together with the operational state. It is therefore necessary that "config false" data is in a different branch than the "config true" data if the operational state can have a different lifetime compared to configuration or if configuration is not immediately or successfully applied.

- Several implementations have proprietary mechanisms that allow clients to store inactive data in <running>. Inactive data is conceptually removed before validation.

- Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.

- 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.

5. Architectural Model of Datastores

Below is a new conceptual model of datastores extending the original model in order to reflect the experience gained with the original model.

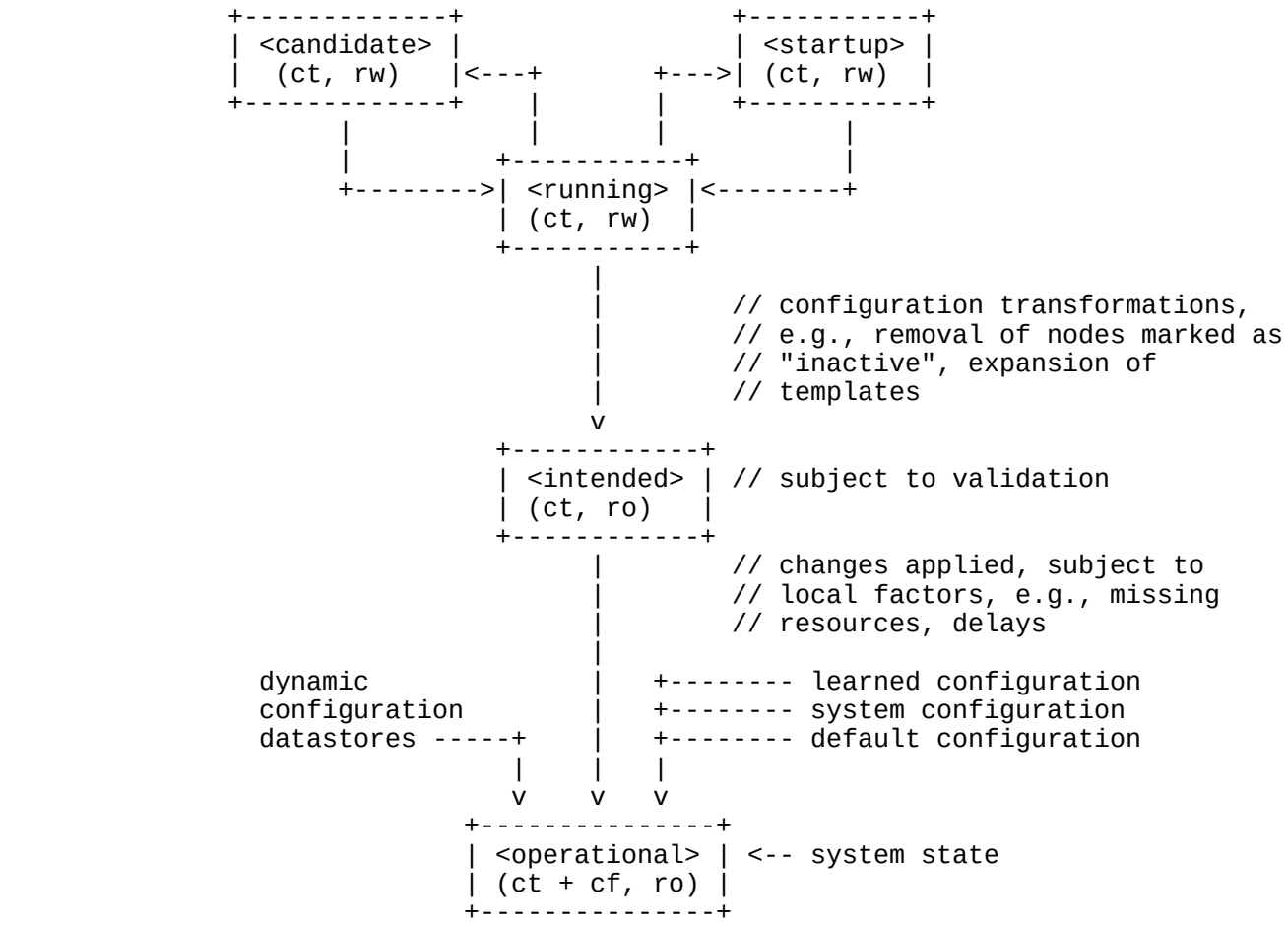


Figure 2

5.1. Conventional Configuration Datastores

The conventional configuration datastores are a set of configuration datastores that share exactly the same datastore schema, allowing data to be copied between them. The term is meant as a generic umbrella description of these datastores. If a module does not contain any configuration data nodes and it is not needed to satisfy any imports, then it MAY be omitted from the datastore schema for the

conventional configuration datastores. The set of datastores include:

- <running>
- <candidate>
- <startup>
- <intended>

Other conventional configuration datastores may be defined in future documents.

The flow of data between these datastores is depicted in Section 5.

The specific protocols may define explicit operations to copy between these datastores, e.g., NETCONF defines the <copy-config> operation.

5.1.1. The Startup Configuration Datastore (<startup>)

The startup configuration datastore (<startup>) is a configuration datastore holding the configuration loaded by the device when it boots. <startup> is only present on devices that separate the startup configuration from the running configuration datastore.

The startup configuration datastore may not be supported by all protocols or implementations.

On devices that support non-volatile storage, the contents of <startup> will typically persist across reboots via that storage. At boot time, the device loads the saved startup configuration into <running>. To save a new startup configuration, data is copied to <startup>, either via implicit or explicit protocol operations.

5.1.2. The Candidate Configuration Datastore (<candidate>)

The candidate configuration datastore (<candidate>) is a configuration datastore that can be manipulated without impacting the device’s current configuration and that can be committed to <running>.

The candidate configuration datastore may not be supported by all protocols or implementations.

<candidate> does not typically persist across reboots, even in the presence of non-volatile storage. If <candidate> is stored using

non-volatile storage, it is reset at boot time to the contents of <running>.

5.1.3. The Running Configuration Datastore (<running>)

The running configuration datastore (<running>) is a configuration datastore that holds the current configuration of the device. It MAY include configuration that requires further transformation before it can be applied, e.g., inactive configuration, or template-mechanism-oriented configuration that needs further expansion. However, <running> MUST always be a valid configuration data tree, as defined in Section 8.1 of [RFC7950].

<running> MUST be supported if the device can be configured via conventional configuration datastores.

If a device does not have a distinct <startup> and non-volatile storage is available, the device will typically use that non-volatile storage to allow <running> to persist across reboots.

5.1.4. The Intended Configuration Datastore (<intended>)

The intended configuration datastore (<intended>) is a read-only configuration datastore. It represents the configuration after all configuration transformations to <running> are performed (e.g., template expansion, removal of inactive configuration), and is the configuration that the system attempts to apply.

<intended> is tightly coupled to <running>. Whenever data is written to <running>, then <intended> MUST also be immediately updated by performing all necessary configuration transformations to the contents of <running> and then <intended> is validated.

<intended> MAY also be updated independently of <running> if the effect of a configuration transformation changes, but <intended> MUST always be a valid configuration data tree, as defined in Section 8.1 of [RFC7950].

For simple implementations, <running> and <intended> are identical.

The contents of <intended> are also related to the "config true" subset of <operational>, and hence a client can determine to what extent the intended configuration is currently in use by checking whether the contents of <intended> also appear in <operational>.

<intended> does not persist across reboots; its relationship with <running> makes that unnecessary.

Currently there are no standard mechanisms defined that affect <intended> so that it would have different content than <running>, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>. A second example is support for templates, which can perform transformations on the configuration from <running> to the configuration written to <intended>.

5.2. Dynamic Configuration Datastores

The model recognizes the need for dynamic configuration 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 dynamic configuration datastores interact with the rest of the system through <operational>.

The datastore schema for a dynamic configuration datastore MAY differ from the datastore schema used for conventional configuration datastores. If a module does not contain any configuration data nodes and it is not needed to satisfy any imports, then it MAY be omitted from the datastore schema for the dynamic configuration datastore.

5.3. The Operational State Datastore (<operational>)

The operational state datastore (<operational>) is a read-only datastore that consists of all "config true" and "config false" nodes defined in the datastore’s schema. In the original NETCONF 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.

<operational> contains system state and all configuration actually used by the system. This includes all applied configuration from <intended>, learned configuration, system-provided configuration, and default values defined by any supported data models. In addition, <operational> also contains applied configuration from dynamic configuration datastores.

The datastore schema for <operational> MUST be a superset of the combined datastore schema used in all configuration datastores except that configuration data nodes supported in a configuration datastore MAY be omitted from <operational> if a server is not able to accurately report them.

Requests to retrieve nodes from <operational> always return the value in use if the node exists, regardless of any default value specified in the YANG module. If no value is returned for a given node, then this implies that the node is not used by the device.

The interpretation of what constitutes as being "in use" by the system is dependent on both the schema definition and the device implementation. Generally, functionality that is enabled and operational on the system would be considered as being "in use". Conversely, functionality that is neither enabled nor operational on the system is considered as not being "in use", and hence SHOULD be omitted from <operational>.

<operational> SHOULD conform to any constraints specified in the data model, but given the principal aim of returning "in use" values, it is possible that constraints MAY be violated under some circumstances, e.g., an abnormal value is "in use", the structure of a list is being modified, or due to remnant configuration (see Section 5.3.1). Note, that deviations SHOULD be used when it is known in advance that a device does not fully conform to the <operational> schema.

Only semantic constraints MAY be violated, these are the YANG "when", "must", "mandatory", "unique", "min-elements", and "max-elements" statements; and the uniqueness of key values.

Syntactic constraints MUST NOT be violated, including hierarchical organization, identifiers, and type-based constraints. If a node in <operational> does not meet the syntactic constraints then it MUST NOT be returned, and some other mechanism should be used to flag the error.

<operational> does not persist across reboots.

5.3.1. Remnant Configuration

Changes to configuration may take time to percolate through to <operational>. During this period, <operational> may contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device. Such remnant configuration from the previous configuration persists until the system has released resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles).

Remnant configuration is a common example of where the semantic constraints defined in the data model cannot be relied upon for <operational>, since the system may have remnant configuration whose constraints were valid with the previous configuration and that are

not valid with the current configuration. Since constraints on "config false" nodes may refer to "config true" nodes, remnant configuration may force the violation of those constraints.

5.3.2. Missing Resources

Configuration in <intended> can refer to resources that are not available or otherwise not physically present. In these situations, these parts of <intended> are not applied. The data appears in <intended> but does not appear in <operational>.

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 <operational>.

Note that configuration validity cannot depend on the current state of such resources, since that would imply that removing a resource might render the configuration invalid. This is unacceptable, especially given that rebooting such a device would cause it to restart with an invalid configuration. Instead we allow configuration for missing resources to exist in <running> and <intended>, but it will not appear in <operational>.

5.3.3. System-controlled Resources

Sometimes resources are controlled by the device and the corresponding system controlled data appears in (and disappears from) <operational> 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 <operational> eventually (if application of the configuration was successful).

5.3.4. Origin Metadata Annotation

As configuration flows into <operational>, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The origin applies to all configuration nodes except non-presence containers. The "origin" metadata annotation is defined in Section 7. The values are YANG identities. The following identities are defined:

- origin: abstract base identity from which the other origin identities are derived.

- intended: represents configuration provided by <intended>.

- dynamic: represents configuration provided by a dynamic configuration datastore.

- system: represents configuration provided by the system itself. Examples of system configuration include applied configuration for an always existing loopback interface, or interface configuration that is auto-created due to the hardware currently present in the device.

- learned: represents configuration that has been learned via protocol interactions with other systems, including protocols such as link-layer negotiations, routing protocols, DHCP, etc.

- default: represents configuration using a default value specified in the data model, using either values in the "default" statement or any values described in the "description" statement. The default origin is only used when the configuration has not been provided by any other source.

- unknown: represents configuration for which the system cannot identify the origin.

These identities can be further refined, e.g., there could be separate identities for particular types or instances of dynamic configuration datastores derived from "dynamic".

For all configuration data nodes in <operational>, the device SHOULD report the origin that most accurately reflects the source of the configuration that is in use by the system.

In cases where it could be ambiguous as to which origin should be used, i.e. where the same data node value has originated from multiple sources, then the description statement in the YANG module SHOULD be used as guidance for choosing the appropriate origin. For example:

If for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value overrides any configured value, then the origin would be reported as "learned", even when a learned value is the same as the configured value.

Conversely, if for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value does not override an explicitly configured value, then the origin would be reported as "intended" even when a learned value is the same as the configured value.

In the case that a device cannot provide an accurate origin for a particular configuration data node then it SHOULD use the origin "unknown".

6. Implications on YANG

6.1. XPath Context

This section updates section 6.4.1 of RFC 7950.

If a server implements the architecture defined in this document, the accessible trees for some XPath contexts are refined as follows:

- If the XPath expression is defined in a substatement to a data node that represents system state, the accessible tree is all operational state in the server. The root node has all top-level data nodes in all modules as children.

- If the XPath expression is defined in a substatement to a "notification" statement, the accessible tree is the notification instance and all operational state in the server. If the notification is defined on the top level in a module, then the root node has the node representing the notification being defined and all top-level data nodes in all modules as children. Otherwise, the root node has all top-level data nodes in all modules as children.

- If the XPath expression is defined in a substatement to an "input" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation’s input parameters as children.

- If the XPath expression is defined in a substatement to an "output" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation’s output parameters as children.

6.2. Invocation of Actions and RPCs

This section updates section 7.15 of RFC 7950.

Actions are always invoked in the context of the operational state datastore. The node for which the action is invoked MUST exist in the operational state datastore.

Note that this document does not constrain the result of invoking an RPC or action in any way. For example, an RPC might be defined to modify the contents of some datastore.

7. YANG Modules

<CODE BEGINS> file "ietf-datastores@2018-01-11.yang"

```
module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

  organization
    "IETF Network Modeling (NETMOD) Working Group";

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

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

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

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

     Author:   Phil Shafer
               <mailto:phil@juniper.net>

     Author:   Kent Watsen
               <mailto:kwatsen@juniper.net>

     Author:   Rob Wilton
               <rwilton@cisco.com>";

  description
    "This YANG module defines two sets of identities for datastores.
     The first identifies the datastores themselves, the second
     identifies datastore properties.
```

```
        Copyright (c) 2018 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
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

    revision 2018-01-11 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: Network Management Datastore Architecture";
    }

    /*
     * Identities
     */

    identity datastore {
      description
        "Abstract base identity for datastore identities.";
    }

    identity conventional {
      base datastore;
      description
        "Abstract base identity for conventional configuration
         datastores.";
    }

    identity running {
      base conventional;
      description
        "The running configuration datastore.";
    }

    identity candidate {
      base conventional;
      description
        "The candidate configuration datastore.";
    }
```

```
    identity startup {
      base conventional;
      description
        "The startup configuration datastore.";
    }

    identity intended {
      base conventional;
      description
        "The intended configuration datastore.";
    }

    identity dynamic {
      base datastore;
      description
        "Abstract base identity for dynamic configuration datastores.";
    }

    identity operational {
      base datastore;
      description
        "The operational state datastore.";
    }

    /*
     * Type definitions
     */

    typedef datastore-ref {
      type identityref {
        base datastore;
      }
      description
        "A datastore identity reference.";
    }

  }

  <CODE ENDS>

  <CODE BEGINS> file "ietf-origin@2018-01-11.yang"

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

    import ietf-yang-metadata {
```

```
      prefix md;
    }

    organization
      "IETF Network Modeling (NETMOD) Working Group";

    contact
      "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

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

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

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

       Author:   Phil Shafer
                 <mailto:phil@juniper.net>

       Author:   Kent Watsen
                 <mailto:kwatsen@juniper.net>

       Author:   Rob Wilton
                 <rwilton@cisco.com>";

    description
      "This YANG module defines an 'origin' metadata annotation, and a
       set of identities for the origin value.

       Copyright (c) 2018 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
       (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
       for full legal notices.";

    revision 2018-01-11 {
      description
        "Initial revision.";
      reference
```

```
        "RFC XXXX: Network Management Datastore Architecture";
    }

    /*
     * Identities
     */

    identity origin {
      description
        "Abstract base identity for the origin annotation.";
    }

    identity intended {
      base origin;
      description
        "Denotes configuration from the intended configuration
         datastore";
    }

    identity dynamic {
      base origin;
      description
        "Denotes configuration from a dynamic configuration
         datastore.";
    }

    identity system {
      base origin;
      description
        "Denotes configuration originated by the system itself.

         Examples of system configuration include applied configuration
         for an always existing loopback interface, or interface
         configuration that is auto-created due to the hardware
         currently present in the device.";
    }

    identity learned {
      base origin;
      description
        "Denotes configuration learned from protocol interactions with
         other devices, instead of via either the intended
         configuration datastore or any dynamic configuration
         datastore.

         Examples of protocols that provide learned configuration
         include link-layer negotiations, routing protocols, and
         DHCP.";
```

```
    }

    identity default {
      base origin;
      description
        "Denotes configuration that does not have an configured or
         learned value, but has a default value in use.  Covers both
         values defined in a 'default' statement, and values defined
         via an explanation in a 'description' statement.";
    }

    identity unknown {
      base origin;
      description
        "Denotes configuration for which the system cannot identify the
         origin.";
    }

    /*
     * Type definitions
     */

    typedef origin-ref {
      type identityref {
        base origin;
      }
      description
        "An origin identity reference.";
    }

    /*
     * Metadata annotations
     */

    md:annotation origin {
      type origin-ref;
      description
        "The 'origin' annotation can be present on any configuration
         data node in the operational state datastore.  It specifies
         from where the node originated.  If not specified for a given
         configuration data node then the origin is the same as the
         origin of its parent node in the data tree.  The origin for
         any top level configuration data nodes must be specified.";
    }
  }

  <CODE ENDS>
```

8. IANA Considerations

8.1. Updates to the IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-datastores
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-origin
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
```

8.2. Updates to the YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

```
name:         ietf-datastores
namespace:    urn:ietf:params:xml:ns:yang:ietf-datastores
prefix:       ds
reference:    RFC XXXX

name:         ietf-origin
namespace:    urn:ietf:params:xml:ns:yang:ietf-origin
prefix:       or
reference:    RFC XXXX
```

9. Security Considerations

This document discusses an architectural model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

Although this document specifies several YANG modules, these modules only define identities and a metadata annotation, hence the "YANG module security guidelines" do not apply.

The origin metadata annotation exposes the origin of values in the applied configuration. Origin information may provide hints that certain control plane protocols are active on a device. Since origin information is tied to applied configuration values, it is only accessible to clients that have the permissions to read the applied configuration values. Security administrators should consider the

sensitivity of origin information while defining access control rules.

10. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([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:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Andy Bierman, YumaWorks, <andy@yumaworks.com>
- Marcus Hines, Google, <hines@google.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Balazs Lengyel, Ericsson, <balazs.lengyel@ericsson.com>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Tom Petch, Engineering Networks Ltd, <ietfc@btconnect.com>
- Anees Shaikh, Google, <aashaikh@google.com>
- Rob Shakir, Google, <robjs@google.com>
- Jason Sterne, Nokia, <jason.sterne@nokia.co>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

11. References

11.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

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

[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>.

11.2. 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.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <https://www.rfc-editor.org/info/rfc6244>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <https://www.rfc-editor.org/info/rfc7277>.

Appendix A. Guidelines for Defining Datastores

The definition of a new datastore in this architecture should be provided in a document (e.g., an RFC) purposed to the definition of the datastore. When it makes sense, more than one datastore may be defined in the same document (e.g., when the datastores are logically connected). Each datastore's definition should address the points specified in the sections below.

A.1. Define which YANG modules can be used in the datastore

Not all YANG modules may be used in all datastores. Some datastores may constrain which data models can be used in them. If it is desirable that a subset of all modules can be targeted to the datastore, then the documentation defining the datastore must indicate this.

A.2. Define which subset of YANG-modeled data applies

By default, the data in a datastore is modeled by all YANG statements in the available YANG modules. However, it is possible to specify criteria that YANG statements must satisfy in order to be present in a datastore. For instance, maybe only "config true" nodes, or "config false" nodes that also have a specific YANG extension, are present in the datastore.

A.3. Define how data is actualized

The new datastore must specify how it interacts with other datastores.

For example, the diagram in Section 5 depicts dynamic configuration datastores feeding into <operational>. How this interaction occurs has to be defined by the particular dynamic configuration datastores. In some cases, it may occur implicitly, as soon as the data is put into the dynamic configuration datastore while, in other cases, an explicit action (e.g., an RPC) may be required to trigger the application of the datastore’s data.

A.4. Define which protocols can be used

By default, it is assumed that both the NETCONF and RESTCONF protocols can be used to interact with a datastore. However, it may be that only a specific protocol can be used (e.g., ForCES) or that a subset of all protocol operations or capabilities are available (e.g., no locking or no XPath-based filtering).

A.5. Define YANG identities for the datastore

The datastore must be defined with a YANG identity that uses the "ds:datastore" identity, or one of its derived identities, as its base. This identity is necessary so that the datastore can be referenced in protocol operations (e.g., <get-data>).

The datastore may also be defined with an identity that uses the "or:origin" identity or one its derived identities as its base. This identity is needed if the datastore interacts with <operational> so that data originating from the datastore can be identified as such via the "origin" metadata attribute defined in Section 7.

An example of these guidelines in use is provided in Appendix B.

Appendix B. Ephemeral Dynamic Configuration Datastore Example

The section defines documentation for an example dynamic configuration datastore using the guidelines provided in Appendix A. While this example is very terse, it is expected to be that a standalone RFC would be needed when fully expanded.

This example defines a dynamic configuration datastore called "ephemeral", which is loosely modeled after the work done in the I2RS working group.

Name	Value
Name	ephemeral
YANG modules	all (default)
YANG nodes	all "config true" data nodes
How applied	changes automatically propagated to <operational>
Protocols	NC/RC (default)
YANG Module	(see below)

The example "ephemeral" datastore properties

```
   module example-ds-ephemeral {
     yang-version 1.1;
     namespace "urn:example:ds-ephemeral";
     prefix eph;

     import ietf-datastores {
       prefix ds;
     }
     import ietf-origin {
       prefix or;
     }

     // datastore identity
     identity ds-ephemeral {
       base ds:dynamic;
       description
         "The ephemeral dynamic configuration datastore.";
     }

     // origin identity
     identity or-ephemeral {
       base or:dynamic;
       description
         "Denotes data from the ephemeral dynamic configuration
          datastore.";
     }
   }
```

Appendix C. Example Data

The use of datastores is complex, and many of the subtle effects are more easily presented using examples. This section presents a series of example data models with some sample contents of the various datastores.

C.1. System Example

In this example, the following fictional module is used:

```
   module example-system {
     yang-version 1.1;
     namespace urn:example:system;
     prefix sys;

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

```
     container system {
       leaf hostname {
         type string;
       }

       list interface {
         key name;

         leaf name {
           type string;
         }

         container auto-negotiation {
           leaf enabled {
             type boolean;
             default true;
           }
           leaf speed {
             type uint32;
             units mbps;
             description
               "The advertised speed, in mbps.";
           }
         }

         leaf speed {
           type uint32;
           units mbps;
           config false;
           description
             "The speed of the interface, in mbps.";
         }

         list address {
           key ip;

           leaf ip {
             type inet:ip-address;
           }
           leaf prefix-length {
             type uint8;
           }
         }
       }
     }
   }
```

The operator has configured the host name and two interfaces, so the contents of <intended> are:

```
<system xmlns="urn:example:system">

  <hostname>foo.example.com</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

  <interface>
    <name>eth1</name>
    <address>
      <ip>2001:db8::20</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

</system>
```

The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. In addition to a default value, a loopback interface is automatically added by the system, and the result of the "speed" auto-negotiation. All of this is reflected in <operational>. Note how the origin metadata attribute for several "config true" data nodes is inherited from their parent data nodes.

```
   <system
       xmlns="urn:example:system"
       xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

     <hostname or:origin="or:learned">bar.example.com</hostname>

     <interface or:origin="or:intended">
       <name>eth0</name>
       <auto-negotiation>
         <enabled or:origin="or:default">true</enabled>
         <speed>1000</speed>
       </auto-negotiation>
       <speed>100</speed>
       <address>
         <ip>2001:db8::10</ip>
         <prefix-length>64</prefix-length>
       </address>
       <address or:origin="or:learned">
         <ip>2001:db8::1:100</ip>
         <prefix-length>64</prefix-length>
       </address>
     </interface>

     <interface or:origin="or:system">
       <name>lo0</name>
       <address>
         <ip>::1</ip>
         <prefix-length>128</prefix-length>
       </address>
     </interface>

   </system>
```

C.2. BGP Example

Consider the following fragment of a fictional BGP module:

```
      container bgp {
        leaf local-as {
          type uint32;
        }
        leaf peer-as {
          type uint32;
        }
        list peer {
          key name;
          leaf name {
            type inet:ip-address;
          }
          leaf local-as {
            type uint32;
            description
              ".... Defaults to ../local-as";
          }
          leaf peer-as {
            type uint32;
            description
               "... Defaults to ../peer-as";
          }
          leaf local-port {
            type inet:port;
          }
          leaf remote-port {
            type inet:port;
            default 179;
          }
          leaf state {
            config false;
            type enumeration {
              enum init;
              enum established;
              enum closing;
            }
          }
        }
      }
```

In this example model, both bgp/peer/local-as and bgp/peer/peer-as have complex hierarchical values, allowing the user to specify default values for all peers in a single location.

The model also follows the pattern of fully integrating state ("config false") nodes with configuration ("config true") nodes. There is no separate "bgp-state" hierarchy, with the accompanying

repetition of containment and naming nodes. This makes the model simpler and more readable.

C.2.1. Datastores

Each datastore represents differing views of these nodes. <running> will hold the configuration provided by the operator, for example a single BGP peer. <intended> will conceptually hold the data as validated, after the removal of data not intended for validation and after any local template mechanisms are performed. <operational> will show data from <intended> as well as any "config false" nodes.

C.2.2. Adding a Peer

If the user configures a single BGP peer, then that peer will be visible in both <running> and <intended>. It may also appear in <candidate>, if the server supports the candidate configuration datastore. Retrieving the peer will return only the user-specified values.

No time delay should exist between the appearance of the peer in <running> and <intended>.

In this scenario, we've added the following to <running>:

```
<bgp>
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
  </peer>
</bgp>
```

C.2.2.1. <operational>

The operational datastore will contain the fully expanded peer data, including "config false" nodes. In our example, this means the "state" node will appear.

In addition, <operational> will contain the "currently in use" values for all nodes. This means that local-as and peer-as will be populated even if they are not given values in <intended>. The value of bgp/local-as will be used if bgp/peer/local-as is not provided; bgp/peer-as and bgp/peer/peer-as will have the same relationship. In the operational view, this means that every peer will have values for their local-as and peer-as, even if those values are not explicitly configured but are provided by bgp/local-as and bgp/peer-as.

Each BGP peer has a TCP connection associated with it, using the values of local-port and remote-port from <intended>. If those values are not supplied, the system will select values. When the connection is established, <operational> will contain the current values for the local-port and remote-port nodes regardless of the origin. If the system has chosen the values, the "origin" attribute will be set to "system". Before the connection is established, one or both of the nodes may not appear, since the system may not yet have their values.

```
<bgp or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>established</state>
  </peer>
</bgp>
```

C.2.3. Removing a Peer

Changes to configuration may take time to percolate through the various software components involved. During this period, it is imperative to continue to give an accurate view of the working of the device. <operational> will contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device.

Consider the scenario where a client removes a BGP peer. When a peer is removed, the operational state will continue to reflect the existence of that peer until the peer’s resources are released, including closing the peer’s connection. During this period, the current data values will continue to be visible in <operational>, with the "origin" attribute set to indicate the origin of the original data.

```
<bgp or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>closing</state>
  </peer>
</bgp>
```

Once resources are released and the connection is closed, the peer's data is removed from <operational>.

C.3. Interface Example

In this section, we will use this simple interface data model:

```
container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
    }
  }
}
```

C.3.1. Pre-provisioned Interfaces

One common issue in networking devices is the support of Field Replaceable Units (FRUs) that can be inserted and removed from the device without requiring a reboot or interfering with normal operation. These FRUs are typically interface cards, and the devices support pre-provisioning of these interfaces.

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, then <intended> might contain the following:

```
  <interfaces>
    <interface>
      <name>et-0/0/0</name>
      <description>Test interface</description>
    </interface>
  </interfaces>
```

Since the interface does not exist, this data does not appear in <operational>.

When a FRU containing this interface is inserted, the system will detect it and process the associated configuration. <operational> will contain the data from <intended>, as well as nodes added by the system, such as the current value of the interface's MTU.

```
  <interfaces or:origin="or:intended">
    <interface>
      <name>et-0/0/0</name>
      <description>Test interface</description>
      <mtu or:origin="or:system">1500</mtu>
    </interface>
  </interfaces>
```

If the FRU is removed, the interface data is removed from <operational>.

C.3.2. System-provided Interface

Imagine if the system provides a loopback interface (named "lo0") with a default ip-address of "127.0.0.1" and a default ip-address of "::1". The system will only provide configuration for this interface if there is no data for it in <intended>.

When no configuration for "lo0" appears in <intended>, then <operational> will show the system-provided data:

```
  <interfaces or:origin="or:intended">
    <interface or:origin="or:system">
      <name>lo0</name>
      <ip-address>127.0.0.1</ip-address>
      <ip-address>::1</ip-address>
    </interface>
  </interfaces>
```

When configuration for "lo0" does appear in <intended>, then <operational> will show that data with the origin set to "intended". If the "ip-address" is not provided, then the system-provided value will appear as follows:

```
  <interfaces or:origin="or:intended">
    <interface>
      <name>lo0</name>
      <description>loopback</description>
      <ip-address or:origin="or:system">127.0.0.1</ip-address>
      <ip-address>::1</ip-address>
    </interface>
  </interfaces>
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2017

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

March 6, 2017

YANG Schema Mount
draft-ietf-netmod-schema-mount-04

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 September 7, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
2. Terminology and Notation 5
2.1. Glossary of New Terms 6
2.2. Tree Diagrams 6
2.3. Namespace Prefixes 6
3. Schema Mount 7
3.1. Mount Point Definition 7
3.2. Specification of the Mounted Schema 7
3.3. Multiple Levels of Schema Mount 11
4. Refering to Data Nodes in the Parent Schema 11
5. RPC operations and Notifications 12
6. Implementation Notes 13
7. Data Model 13
8. Schema Mount YANG Module 15
9. IANA Considerations 21
10. Security Considerations 21
11. Contributors 21
12. References 21
12.1. Normative References 21
12.2. Informative References 22
Appendix A. Example: Device Model with LNEs and NIs 23
A.1. Physical Device 23
A.2. Logical Network Elements 24
A.3. Network Instances 27
A.4. Invoking an RPC Operation 29
Appendix B. Open Issues 29
B.1. Referencing Mount Points Using Schema Node Identifiers . 29
B.2. Defining the "mount-point" Extension in a Separate Module 30
B.3. Parent References 31
B.4. RPC Operations and Notifications in Mounted Modules . . . 31
B.5. Tree Representation 32
B.6. Design-Time Mounts 32
Authors’ Addresses 32

1. Introduction

Modularity and extensibility were among the leading design principles of the YANG data modeling language. As a result, the same YANG module can be combined with various sets of other modules and thus form a data model that is tailored to meet the requirements of a specific use case. Server implementors are only required to specify all YANG modules comprising the data model (together with their revisions and other optional choices) in the YANG library data ([RFC7895], and Section 5.6.4 of [RFC7950]) implemented by the server. Such YANG modules appear in the data model "side by side",

i.e., top-level data nodes of each module - if there are any - are also top-level nodes of the overall data model.

Furthermore, YANG has two mechanisms for contributing a schema hierarchy defined elsewhere to the contents of an internal node of the schema tree; these mechanisms are realized through the following YANG statements:

- The "uses" statement explicitly incorporates the contents of a grouping defined in the same or another module. See Section 4.2.6 of [RFC7950] for more details.

- The "augment" statement explicitly adds contents to a target node defined in the same or another module. See Section 4.2.8 of [RFC7950] for more details.

With both mechanisms, the source or target YANG module explicitly defines the exact location in the schema tree where the new nodes are placed.

In some cases these mechanisms are not sufficient; it is often necessary that an existing module (or a set of modules) is added to the data model starting at a non-root location. For example, YANG modules such as "ietf-interfaces" [RFC7223] are often defined so as to be used in a data model of a physical device. Now suppose we want to model a device that supports multiple logical devices [I-D.ietf-rtgwg-lne-model], each of which has its own instantiation of "ietf-interfaces", and possibly other modules, but, at the same time, we want to be able to manage all these logical devices from the master device. Hence, we would like to have a schema like this:

```
  +--rw interfaces
  |  +--rw interface* [name]
  |     ...
  +--rw logical-device* [name]
     +--rw name
     |   ...
     +--rw interfaces
       +--rw interface* [name]
          ...
```

With the "uses" approach, the complete schema tree of "ietf-interfaces" would have to be wrapped in a grouping, and then this grouping would have to be used at the top level (for the master device) and then also in the "logical-device" list (for the logical devices). This approach has several disadvantages:

- It is not scalable because every time there is a new YANG module that needs to be added to the logical device model, we have to update the model for logical devices with another "uses" statement pulling in contents of the new module.

- Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.

- Nodes defined inside a grouping belong to the namespace of the module where it is used, which makes references to such nodes from other modules difficult or even impossible.

- It would be difficult for vendors to add proprietary modules when the "uses" statements are defined in a standard module.

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 at the top level. The same hierarchy of nodes would thus have to be defined twice, which is clearly not scalable either.

This document introduces a new generic mechanism, denoted as schema mount, that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Unlike the "uses" and "augment" approaches discussed above, the mounted modules needn't be specially prepared for mounting and, consequently, existing modules such as "ietf-interfaces" can be mounted without any modifications.

The basic idea of schema mount is to label a data node in the parent schema as the mount point, and then define a complete data model to be attached to the mount point so that the labeled data node effectively becomes the root node of the mounted data model.

In principle, the mounted schema can be specified at three different phases of the data model life cycle:

1. Design-time: the mounted schema is defined along with the mount point in the parent module. In this case, the mounted schema has to be the same for every implementation of the parent module.

2. Implementation-time: the mounted schema is defined by a server implementor and is as stable as YANG library information, i.e., it may change after an upgrade of server software but not after rebooting the server. Also, a client can learn the entire schema together with YANG library data.

3. Run-time: the mounted schema is defined by instance data that is part of the mounted data model. If there are multiple instances of the same mount point (e.g., in multiple entries of a list), the mounted data model may be different for each instance.

The schema mount mechanism defined in this document provides support only for the latter two cases because design-time definition of the mounted schema doesn't play well with the existing YANG modularity mechanisms. For example, it would be impossible to augment the mounted data model.

Schema mount applies to the data model, and specifically does not assume anything about the source of instance data for the mounted schemas. 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 such as mounting sub-hierarchies of a module.

2. Terminology and Notation

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].

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

- client
- notification
- server

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

- action
- configuration data
- container
- list

- operation

The following terms are defined in [RFC7223] and are not redefined here:

- system-controlled interface

2.1. Glossary of New Terms

- inline schema: a mounted schema whose definition is provided as part of the mounted data, using YANG library [RFC7895].

- mount point: container or list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines the name of the mount point.

- parent schema (of a particular mounted schema): the schema that contains the mount point for the mounted schema.

- top-level schema: a schema according to [RFC7950] in which schema trees of each module (except augments) start at the root node.

2.2. 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 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.

2.3. Namespace Prefixes

In this document, names of data nodes, YANG extensions, 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
yangmnt	ietf-yang-schema-mount	Section 8
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895]

Table 1: Namespace Prefixes

3. Schema Mount

The schema mount mechanism defined in this document provides a new extensibility mechanism for use with YANG 1.1. In contrast to the existing mechanisms described in Section 1, schema mount defines the relationship between the source and target YANG modules outside these modules. The procedure consists of two separate steps that are described in the following subsections.

3.1. Mount Point Definition

A "container" or "list" node becomes a mount point if the "mount-point" extension (defined in the "ietf-yang-schema-mount" module) is used in its definition. This extension can appear only as a substatement of "container" and "list" statements.

The argument of the "mount-point" extension is a YANG identifier that defines the name of the mount point. A module MAY contain multiple "mount-point" statements having the same argument.

It is therefore up to the designer of the parent schema to decide about the placement of mount points. A mount point can also be made conditional by placing "if-feature" and/or "when" as substatements of the "container" or "list" statement that represents the mount point.

The "mount-point" statement MUST NOT be used in a YANG version 1 module. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

3.2. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are defined as state data in the "yangmnt:schema-mounts" container. Data in this container is intended to be as stable as data in the top-level YANG

library [RFC7895]. In particular, it SHOULD NOT change during the same management session.

The "schema-mount" container has the "mount-point" list as one of its children. Every entry of this list refers through its key to a mount point and specifies the mounted schema.

If a mount point is defined in the parent schema but does not have an entry in the "mount-point" list, then the mounted schema is void, i.e., instances of that mount point MUST NOT contain any data above those that are defined in the parent schema.

If multiple mount points with the same name are defined in the same module - either directly or because the mount point is defined in a grouping and the grouping is used multiple times - then the corresponding "mount-point" entry applies equally to all such mount points.

The "config" property of mounted schema nodes is overriden and all nodes in the mounted schema are read-only ("config false") if at least one of the following conditions is satisfied for a mount point:

1. The mount point is itself defined as "config false".

2. The "config" leaf in the corresponding entry of the "mount-point" list is set to "false".

An entry of the "mount-point" list can specify the mounted schema in two different ways:

1. by stating that the schema is available inline, i.e., in run-time instance data; or

2. by referring to one or more entries of the "schema" list in the same instance of "schema-mounts".

In case 1, every instance of the mount point that exists in the parent tree MUST contain a copy of YANG library data [RFC7895] that defines the mounted schema exactly as for a top-level data model. A client is expected to retrieve this data from the instance tree, possibly after creating the mount point. Instances of the same mount point MAY use different mounted schemas.

In case 2, the mounted schema is defined by the combination of all "schema" entries referred to in the "use-schema" list. Optionally, a reference to a "schema" entry can be made conditional by including the "when" leaf. Its argument is an XPath expression that is evaluated in the parent tree with the mount point instance as the

context node. The conditional "schema" entry is used only if the XPath expression evaluates to true. XPath expressions in the argument of "when" may use namespace prefixes that are declared in the "namespace" list (child of "schema-mounts").

Conditional schemas may be used, for example, in a situation where virtual devices are of several different types and the schema for each type is fixed and known in advance. The list of virtual devices in a parent schema module (say "example-virtual-host") might be defined as follows:

```
  list virtual-device {
    key name;
    leaf name {
      type string;
    }
    leaf type {
      type identityref {
        base virtual-device-type;
      }
    }
    container root {
      yangmnt:mount-point virtual-device;
    }
```

The "schema-mounts" specification in state data might contain, for example,

```
    "yangmnt:schema-mounts": {
      "namespace": [
        {
          "prefix": "evh",
          "ns-uri": "http://example.org/ns/example-virtual-host"
        }
      ],
      "mount-point": [
        {
          "module": "example-virtual-host",
          "name": "root",
          "use-schema": [
            {
              "name": "virtual-router-schema",
              "when": "derived-from(../evh:type, 'evh:virtual-router')"
            },
            {
              "name": "virtual-switch-schema",
              "when": "derived-from(../evh:type, 'evh:virtual-switch')"
            }
      ],
      "schema": [
        {
          "name": "virtual-router-schema",
          "module": [
            ...
          ]
        },
        {
          "name": "virtual-switch-schema",
          "module": [
            ...
          ]
        }
      ]
    }
```

The schema of virtual device instances can then be controlled by setting the "type" leaf to an appropriate identity derived from the "virtual-device-type" base.

In case 2, the mounted schema is specified as implementation-time data that can be retrieved together with YANG library data for the parent schema, i.e., even before any instances of the mount point exist. However, the mounted schema has to be the same for all instances of the mount point (except for parts that are conditional due to "when" leaves).

Each entry of the "schema" list contains

- a list in the YANG library format specifying all YANG modules (and revisions etc.) that are implemented or imported in the mounted schema;

- (optionally) a new "schema-mounts" specification that applies to mount points defined within the mounted schema.

3.3. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which subschemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of schema levels. A subschema for a mount point contained in a mounted module can be specified in one of the following ways:

- by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the subschemas exactly as it is done in the top-level schema

- by using the "mount-point" list inside the coresponding "schema" entry.

The former method is applicable to both "inline" and "use-schema" cases whereas the latter requires the "use-schema" case. On the other hand, the latter method allows for a compact representation of a multi-level schema the does not rely on the presence of any instance data.

4. Refering to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted data model works exactly as a top-level data model, i.e., it is confined to the "mount jail". This means that all paths in the mounted data model (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) are interpreted with the mount point as the root node. YANG modules of the mounted schema as well as corresponding instance data thus cannot refer to schema nodes or instance data outside the mount jail.

However, this restriction is sometimes too severe. A typical example are network instances (NI) [I-D.ietf-rtgwg-ni-model], where each NI has its own routing engine but the list of interfaces is global and shared by all NIs. If we want to model this organization with the NI schema mounted using schema mount, the overall schema tree would look schematically as follows:

```
+--rw interfaces
|  +--rw interface* [name]
|     ...
+--rw network-instances
   +--rw network-instance* [name]
      +--rw name
      +--rw root
         +--rw routing
            ...
```

Here, the "root" node is the mount point for the NI schema. Routing configuration inside an NI often needs to refer to interfaces (at least those that are assigned to the NI), which is impossible unless such a reference can point to a node in the parent schema (interface name).

Therefore, schema mount also allows for such references, albeit in a limited and controlled way. The "schema-mounts" container has a child leaf-list named "parent-reference" that contains zero or more module names. All modules appearing in this leaf-list MUST be implemented in the parent schema and MUST NOT be implemented in the mounted schema. All absolute leafref paths and instance identifiers within the mounted data model and corresponding instance data tree are then evaluated as follows:

- If the leftmost node-identifier (right after the initial slash) belongs to the namespace of a module that is listed in "parent-reference", then the root of the accessible tree is not the mount point but the root of the parent schema.

- Other rules for the "leafref" and "instance-identifier" types as defined in Sections 9.9 and 9.13 of [RFC7950] remain in effect.

It is worth emphasizing that the mount jail can be escaped only via absolute leafref paths and instance identifiers. Relative leafref paths, "must"/"when" expressions and schema node identifiers are still restricted to the mounted schema.

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation by representing it as an action defined for the corresponding mount point, see Section 7.15 of ^RFC7950. An example of this is given in Appendix A.4.

Similarly, if the server emits a notification defined at the top level of any mounted module, it MUST be represented as if the

notification was connected to the mount point, see Section 7.16 of [RFC7950].

6. Implementation Notes

Network management of devices that use a data model with schema mount can be implemented in different ways. However, the following implementations options are envisioned as typical:

- shared management: instance data of both parent and mounted schemas are accessible within the same management session.

- split management: one (master) management session has access to instance data of both parent and mounted schemas but, in addition, an extra session exists for every instance of the mount point, having access only to the mounted data tree.

7. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
   module: ietf-yang-schema-mount
       +--ro schema-mounts
          +--ro namespace* [prefix]
          |  +--ro prefix    yang:yang-identifier
          |  +--ro ns-uri?   inet:uri
          +--ro mount-point* [module name]
          |  +--ro module        yang:yang-identifier
          |  +--ro name          yang:yang-identifier
          |  +--ro config?       boolean
          |  +--ro (schema-ref)?
          |     +--:(inline)
          |     |  +--ro inline?       empty
          |     +--:(use-schema)
          |        +--ro use-schema* [name]
          |           +--ro name
          |           |       -> /schema-mounts/schema/name
          |           +--ro when?               yang:xpath1.0
          |           +--ro parent-reference*   yang:yang-identifier
          +--ro schema* [name]
             +--ro name           string
             +--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
             +--ro mount-point* [module name]
                +--ro module        yang:yang-identifier
                +--ro name          yang:yang-identifier
                +--ro config?       boolean
                +--ro (schema-ref)?
                   +--:(inline)
                   |  +--ro inline?       empty
                   +--:(use-schema)
                      +--ro use-schema* [name]
                         +--ro name
                         |       -> /schema-mounts/schema/name
                         +--ro when?               yang:xpath1.0
                         +--ro parent-reference*   yang:yang-identifier
```

8. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

<CODE BEGINS> file "ietf-yang-schema-mount@2017-03-06.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-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  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:   <https://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

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

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

  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 operational state data that specify the
     overall structure of the data model.
```

```
        Copyright (c) 2017 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 2017-03-06 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Schema Mount";
    }

    /*
     * Extensions
     */

    extension mount-point {
      argument name;
      description
        "The argument 'name' is a YANG identifier, i.e., it is of the
         type 'yang:yang-identifier'.

         The 'mount-point' statement MUST NOT be used in a YANG
         version 1 module, neither explicitly nor via a 'uses'
         statement.

         The 'mount-point' statement MAY be present as a substatement
         of 'container' and 'list', and MUST NOT be present elsewhere.

         If a mount point is defined in a grouping, its name is bound
         to the module where the grouping is used.

         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
          /schema-mounts/mount-point list with detailed information on
          which data models are mounted at each mount point.";
      }

      /*
       * Groupings
       */

      grouping mount-point-list {
        description
          "This grouping is used inside the 'schema-mounts' container and
           inside the 'schema' list.";
        list mount-point {
          key "module name";
          description
            "Each entry of this list specifies a schema for a particular
             mount point.

             Each mount point MUST be defined using the 'mount-point'
             extension in one of the modules listed in the corresponding
             YANG library instance with conformance type 'implement'. The
             corresponding YANG library instance is:

             - standard YANG library state data as defined in RFC 7895,
               if the 'mount-point' list is a child of 'schema-mounts',

             - the contents of the sibling 'yanglib:modules-state'
               container, if the 'mount-point' list is a child of
               'schema'.";
          leaf module {
            type yang:yang-identifier;
            description
              "Name of a module containing the mount point.";
          }
          leaf name {
            type yang:yang-identifier;
            description
              "Name of the mount point defined using the 'mount-point'
               extension.";
          }
          leaf config {
            type boolean;
            default "true";
            description
              "If this leaf is set to 'false', then all data nodes in the
               mounted schema are read-only (config false), regardless of
               their 'config' property.";
```

```
        }
        choice schema-ref {
          description
            "Alternatives for specifying the schema.";
          leaf inline {
            type empty;
            description
              "This leaf indicates that the server has mounted
               'ietf-yang-library' and 'ietf-schema-mount' at the mount
               point, and their instantiation (i.e., state data
               containers 'yanglib:modules-state' and 'schema-mounts')
               provides the information about the mounted schema.";
          }
          list use-schema {
            key "name";
            description
              "Each entry of this list contains a reference to a schema
               defined in the /schema-mounts/schema list. The entry can
               be made conditional by specifying an XPath expression in
               the 'when' leaf.";
            leaf name {
              type leafref {
                path "/schema-mounts/schema/name";
              }
              description
                "Name of the referenced schema.";
            }
            leaf when {
              type yang:xpath1.0;
              description
                "This leaf contains an XPath expression. If it is
                 present, then the current entry applies if and only if
                 the expression evaluates to true.

                 The XPath expression is evaluated once for each
                 instance of the data node containing the mount
                 point for which the 'when' leaf is defined.

                 The XPath expression is evaluated using the rules
                 specified in sec. 6.4 of RFC 7950, with these
                 modifications:

                 - The context node is the data node instance
                   containing the corresponding 'mount-point'
                   statement.

                 - The accessible tree contains only data belonging to
                   the parent schema, i.e., all instances of data
```

```
                    nodes containing the mount points are considered
                    empty.

                  - The set of namespace declarations is the set of all
                    prefix/namespace pairs defined in the
                    /schema-mounts/namespace list. Names without a
                    namespace prefix belong to the same namespace as the
                    context node.";
            }
            leaf-list parent-reference {
              type yang:yang-identifier;
              must "not(/schema-mounts/schema[name=current()/../name]/"
                 + "module[name=current() and conformance-type="
                 + "'implement'])" {
                error-message "Parent references cannot be used for a "
                  + "module implemented in the mounted schema.";
                description
                  "Modules that are used for parent references MUST NOT
                   be implemented in the mounted schema.";
              }
              description
                "Entries of this leaf-list are names of YANG modules.
                 All these modules MUST be implemented in the parent
                 schema.

                 Within the mounted schema and the corresponding data
                 tree, conceptual evaluation of absolute leafref paths
                 and instance identifiers is modified in the following
                 way:

                 If the leftmost node-identifier in an absolute leafref
                 path or instance identifier belongs to a module whose
                 name is listed in 'parent-reference', then the root
                 of the accessible data tree coincides with the root of
                 the parent data tree.";
            }
          }
        }
      }
    }

    /*
     * State data nodes
     */

    container schema-mounts {
      config false;
      description
```

```
        "Contains information about the structure of the overall
         mounted data model implemented in the server.";
      list namespace {
        key "prefix";
        description
          "This list provides a mapping of namespace prefixes that are
           used in XPath expressions of 'when' leafs to the
           corresponding namespace URI references.";
        leaf prefix {
          type yang:yang-identifier;
          description
            "Namespace prefix.";
        }
        leaf ns-uri {
          type inet:uri;
          description
            "Namespace URI reference.";
        }
      }
      uses mount-point-list;
      list schema {
        key "name";
        description
          "Each entry specifies a schema that can be mounted at a mount
           point.  The schema information consists of two parts:

           - an instance of YANG library that defines YANG modules used
             in the schema,

           - mount-point list with content identical to the top-level
             mount-point list (this makes the schema structure
             recursive).";
        leaf name {
          type string;
          description
            "Arbitrary name of the schema entry.";
        }
        uses yanglib:module-list;
        uses mount-point-list;
      }
    }
  }

  <CODE ENDS>
```

9. 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
```

10. Security Considerations

TBD

11. 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]); and Lou Berger and Christian Hopps:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Alexander Clemm, Huawei, <alexander.clemm@huawei.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Jan Medved, Cisco, <jmedved@cisco.com>
- Eric Voit, Cisco, <evoit@cisco.com>

12. References

12.1. 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>.

[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>.

[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>.

12.2. 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-05 (work in progress), September 2016.

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-15 (work in progress), February 2017.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-ietf-rtgwg-device-model-01 (work in progress), October 2016.

[I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-01 (work in progress), October 2016.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Network Instances", draft-ietf-rtgwg-ni-model-01 (work in progress), October 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>.

Appendix A. Example: Device Model with LNEs and NIs

This non-normative example demonstrates an implementation of the device model as specified in Section 2 of [I-D.ietf-rtgwg-device-model], using both logical network elements (LNE) and network instances (NI).

A.1. Physical Device

The data model for the physical device may be described by this YANG library content:

```
"ietf-yang-library:modules-state": {
  "module-set-id": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
```

```
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ip",
        "revision": "2014-06-16",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-logical-network-element",
        "revision": "2016-10-21",
        "feature": [
          "bind-lne-name"
        ],
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-library",
        "revision": "2016-06-21",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-schema-mount",
        "revision": "2017-03-06",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-types",
        "revision": "2013-07-15",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
      }
    ]
  }
```

A.2. Logical Network Elements

Each LNE can have a specific data model that is determined at run time, so it is appropriate to mount it using the "inline" method, hence the following "schema-mounts" data:

```
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "name": "root",
      "inline": [null]
    }
  ]
}
```

An administrator of the host device has to configure an entry for each LNE instance, for example,

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-logical-network-element:bind-lne-name": "eth0"
      }
    ]
  },
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "lne-1",
        "managed": true,
        "description": "LNE with NIs",
        "root": {
          ...
        }
      },
      ...
    ]
  }
}
```

and then also place necessary state data as the contents of the "root" instance, which should include at least

o YANG library data specifying the LNE's data model, for example:

```
  "ietf-yang-library:modules-state": {
    "module-set-id": "9358e11874068c8be06562089e94a89e0a392019",
    "module": [
      {
```

```
          "name": "iana-if-type",
          "revision": "2014-05-08",
          "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-inet-types",
          "revision": "2013-07-15",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
          "conformance-type": "import"
        },
        {
          "name": "ietf-interfaces",
          "revision": "2014-05-08",
          "feature": [
            "arbitrary-names",
            "pre-provisioning"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "feature": [
            "ipv6-privacy-autoconf"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-network-instance",
          "revision": "2016-10-27",
          "feature": [
            "bind-network-instance-name"
          ],
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-network-instance",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-schema-mount",
```

```
          "revision": "2017-03-06",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-types",
          "revision": "2013-07-15",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "conformance-type": "import"
        }
      ]
    }
```

o state data for interfaces assigned to the LNE instance (that effectively become system-controlled interfaces for the LNE), for example:

```
  "ietf-interfaces:interfaces-state": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "statistics": {
          "discontinuity-time": "2016-12-16T17:11:27+02:00"
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "fe80::42a8:f0ff:fea8:24fe",
              "origin": "link-layer",
              "prefix-length": 64
            }
          ]
        }
      },
      ...
    ]
  }
```

A.3. Network Instances

Assuming that network instances share the same data model, it can be mounted using the "use-schema" method as follows:

```
    "ietf-yang-schema-mount:schema-mounts": {
      "mount-point": [
        {
          "module": "ietf-network-instance",
          "name": "root",
          "parent-reference": ["ietf-interfaces"],
          "use-schema": [
            {
              "name": "ni-schema"
            }
          ]
        }
      ],
      "schema": [
        {
          "name": "ni-schema",
          "module": [
            {
              "name": "ietf-ipv4-unicast-routing",
              "revision": "2016-11-04",
              "namespace":
               "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
              "conformance-type": "implement"
            },
            {
              "name": "ietf-ipv6-unicast-routing",
              "revision": "2016-11-04",
              "namespace":
               "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
              "conformance-type": "implement"
            },
            {
              "name": "ietf-routing",
              "revision": "2016-11-04",
              "feature": [
                "multiple-ribs",
                "router-id"
              ],
              "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
              "conformance-type": "implement"
            }
          ]
        }
      ]
    }
```

Note also that the "ietf-interfaces" module appears in the "parent-reference" leaf-list for the mounted NI schema. This means

that references to LNE interfaces, such as "outgoing-interface" in static routes, are valid despite the fact that "ietf-interfaces" isn't part of the NI schema.

A.4. Invoking an RPC Operation

Assume that the mounted NI data model also implements the "ietf-isis" module [I-D.ietf-isis-yang-isis-cfg]. An RPC operation defined in this module, such as "clear-adjacency", can be invoked by a client session of a LNE's RESTCONF server as an action tied to a the mount point of a particular network instance using a request URI like this (all on one line):

```
POST /restconf/data/ietf-network-instance:network-instances/
    network-instance=rtrA/root/ietf-isis:clear-adjacency HTTP/1.1
```

Appendix B. Open Issues

B.1. Referencing Mount Points Using Schema Node Identifiers

Each entry in the "mount-point" list is currently identified by two keys, namely YANG module name and mount point name. An alternative is to use a schema node identifier of the mount point as a single key.

For example, the "schema-mounts" data for NI (Appendix A.3) would be changed as follows (the "schema" list doesn't change):

```
"ietf-yang-schema-mount:schema-mounts": {
  "namespace": [
    {
      "prefix": "ni",
      "ns-uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
    }
  ]
  "mount-point": [
    {
      "target": "/ni:network-instances/ni:network-instance/ni:root",
      "parent-reference": ["ietf-interfaces"],
      "use-schema": [
        {
          "name": "ni-schema"
        }
      ]
    }
  ],
  "schema": [
    ...
  ]
}
```

This change would have several advantages:

- the schema mount mechanism becomes even closer to augments, which may simplify implementation
- if a mount point appears inside a grouping, then a different mounted schema can be used for each use of the grouping.
- it optionally allows for use of mount without use of the mount-point extension.

B.2. Defining the "mount-point" Extension in a Separate Module

The "inline" method of schema mounting can be further simplified by defining the "inline" case as the default. That is, if a mount point is defined through the "mount-point" extension but is not present in the "mount-point" list, the "inline" schema mount is assumed.

Consequently, a data model that uses only the "inline" method could omit the "schema-mounts" data entirely, but it still needs to use the "mount-point" extension. In order to enable this, the definition of the "mount-point" extension has to be moved to a YANG module of its own.

A variant of this approach is to completely separate the "inline" and "use-schema" cases by dedicating the "mount-point" extension for use with the "inline" method only (with no "schema-mounts" data), and using schema node identifiers as described in Appendix B.1 for the "use-schema" case.

B.3. Parent References

As explained in Section 4, references to the parent schema can only be used in absolute leafref paths and instance identifiers. However, it is conceivable that they may be useful in other XPath expressions, e.g. in "must" statements. The authors believe it is impossible to allow for parent references in general XPath expressions because, for example, in a location path "//foo:bar" it would be unclear whether the lookup has to be started in the mounted or parent schema.

Should parent references in general XPath be needed, it would be necessary to indicate it explicitly. One way to achieve this is to defining a new XPath function, e.g., parent-root(), that returns the root of the parent data tree.

B.4. RPC Operations and Notifications in Mounted Modules

Turning RPC operations defined in mounted modules into actions tied to the corresponding mount point (see Section 5, and similarly for notifications) is not possible if the path to the mount point in the parent schema contains a keyless list (Section 7.15 of [RFC7950]). The solutions for this corner case are possible:

1. any mount point MUST NOT have a keyless list among its ancestors

2. any mounted module MUST NOT contain RPC operations and/or notifications

3. specifically for each mount point, at least one of the above conditions MUST be satisfied.

4. treat such actions and notifications as non-existing, i.e., ignore them.

The first two requirements seem rather restrictive. On the other hand, the last one is difficult to guarantee - for example, things can break after an augment within the mounted schema.

B.5. Tree Representation

Need to decide how/if mount points are represented in trees.

B.6. Design-Time Mounts

The document currently doesn't provide explicit support for design-time mounts. Design-time mounts have been identified as possibly for multiple cases, and it may be worthwhile to identify a minimum or complete set of modules that must be supported under a mount point. This could be used in service modules that want to allow for configuration of device-specific information. One option could be to add an extension that specify that a certain module is required to be mounted.

Also, if design-time mounts are supported, it could be possible to represent both mounts points and their required modules in tree representations and support for such would need to be defined.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 20, 2019

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

October 17, 2018

YANG Schema Mount
draft-ietf-netmod-schema-mount-12

Abstract

This document defines a mechanism to add the schema trees defined by a set of YANG modules onto a mount point defined in the schema tree in some YANG module.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 20, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
2. Terminology and Notation 5
2.1. Tree Diagrams 6
2.2. Namespace Prefixes 6
3. Schema Mount 7
3.1. Mount Point Definition 7
3.2. Data Model 8
3.3. Specification of the Mounted Schema 8
3.4. Multiple Levels of Schema Mount 9
4. Referring to Data Nodes in the Parent Schema 9
5. RPC operations and Notifications 11
6. Network Management Datastore Architecture (NMDA) Considerations 11
7. Interaction with the Network Configuration Access Control Model (NACM) 11
8. Implementation Notes 12
9. Schema Mount YANG Module 12
10. IANA Considerations 17
11. Security Considerations 17
12. Contributors 18
13. References 19
13.1. Normative References 19
13.2. Informative References 20
Appendix A. Example: Device Model with LNEs and NIs 21
A.1. Physical Device 21
A.2. Logical Network Elements 23
A.3. Network Instances 26
A.4. Invoking an RPC Operation 27
Authors' Addresses 28

1. Introduction

Modularity and extensibility were among the leading design principles of the YANG data modeling language. As a result, the same YANG module can be combined with various sets of other modules and thus form a data model that is tailored to meet the requirements of a specific use case. Server implementors are only required to specify all YANG modules comprising the data model (together with their revisions and other optional choices) in the YANG library data ([RFC7895], [I-D.ietf-netconf-rfc7895bis] and Section 5.6.4 of [RFC7950]) implemented by the server. Such YANG modules appear in the data model "side by side", i.e., top-level data nodes of each module - if there are any - are also top-level nodes of the overall data model.

YANG has two mechanisms for contributing a schema hierarchy defined elsewhere to the contents of an internal node of the schema tree; these mechanisms are realized through the following YANG statements:

- The "uses" statement explicitly incorporates the contents of a grouping defined in the same or another module. See Section 4.2.6 of [RFC7950] for more details.

- The "augment" statement explicitly adds contents to a target node defined in the same or another module. See Section 4.2.8 of [RFC7950] for more details.

With both mechanisms, the YANG module with the "uses" or "augment" statement explicitly defines the exact location in the schema tree where the new nodes are placed.

In some cases these mechanisms are not sufficient; it is sometimes necessary that an existing module (or a set of modules) is added to the data model starting at locations other than the root. For example, YANG modules such as "ietf-interfaces" [RFC8343] are defined so as to be used in a data model of a physical device. Now suppose we want to model a device that supports multiple logical devices [I-D.ietf-rtgwg-lne-model], each of which has its own instantiation of "ietf-interfaces", and possibly other modules, but, at the same time, we want to be able to manage all these logical devices from the master device. Hence, we would like to have a schema tree like this:

```
  +--rw interfaces
  |  +--rw interface* [name]
  |     ...
  +--rw logical-network-element* [name]
     +--rw name
     |   ...
     +--rw interfaces
       +--rw interface* [name]
          ...
```

With the "uses" approach, the complete schema tree of "ietf-interfaces" would have to be wrapped in a grouping, and then this grouping would have to be used at the top level (for the master device) and then also in the "logical-network-element" list (for the logical devices). This approach has several disadvantages:

- It is not scalable because every time there is a new YANG module that needs to be added to the logical device model, we have to update the model for logical devices with another "uses" statement pulling in contents of the new module.

- Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.

- Nodes defined inside a grouping belong to the namespace of the module where it is used, which makes references to such nodes from other modules difficult or even impossible.

- It would be difficult for vendors to add proprietary modules when the "uses" statements are defined in a standard module.

With the "augment" approach, "ietf-interfaces" would have to augment the "logical-network-element" list with all its nodes, and at the same time define all its nodes at the top level. The same hierarchy of nodes would thus have to be defined twice, which is clearly not scalable either.

This document introduces a new mechanism, denoted as schema mount, that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Unlike the "uses" and "augment" approaches discussed above, the mounted modules needn't be specially prepared for mounting and, consequently, existing modules such as "ietf-interfaces" can be mounted without any modifications.

The basic idea of schema mount is to label a data node in the parent schema as the mount point, and then define a complete data model to be attached to the mount point so that the labeled data node effectively becomes the root node of the mounted data model.

In principle, the mounted schema can be specified at three different phases of the data model life cycle:

1. Design-time: the mounted schema is defined along with the mount point in the parent YANG module. In this case, the mounted schema has to be the same for every implementation of the parent module.

2. Implementation-time: the mounted schema is defined by a server implementor and is as stable as the YANG library information of the server.

3. Run-time: the mounted schema is defined by instance data that is part of the mounted data model. If there are multiple instances of the same mount point (e.g., in multiple entries of a list), the mounted data model may be different for each instance.

The schema mount mechanism defined in this document provides support only for the latter two cases. Design-time mounts are outside the

scope of this document, and could be possibly dealt with in a future revision of the YANG data modeling language.

Schema mount applies to the data model, and specifically does not assume anything about the source of instance data for the mounted schemas. 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.

How and when specific mount points are instantiated by the server is out of scope for this document. Such mechanisms may be defined in future specifications.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms such as mounting sub-hierarchies of a module.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342].

2. Terminology and Notation

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

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

- action
- container
- data node
- list
- RPC operation
- schema node
- schema tree

The following terms are defined in [RFC8342] and are not redefined here:

- client
- notification
- operational state
- server

The following term is defined in [RFC8343] and is not redefined here:

- system-controlled interface

The following term is defined in [I-D.ietf-netconf-rfc7895bis] is not redefined here:

- YANG library content identifier

The following additional terms are used within this document:

- mount point: A container or a list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines a label for the mount point.
- schema: A collection of schema trees with a common root.
- top-level schema: A schema rooted at the root node.
- mounted schema: A schema rooted at a mount point.
- parent schema (of a mounted schema): A schema containing the mount point.
- schema mount: The mechanism to combine data models defined in this document.

2.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340]

2.2. Namespace Prefixes

In this document, names of data nodes, YANG extensions, 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
yangmnt	ietf-yang-schema-mount	Section 9
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895], [I-D.ietf-netconf-rfc7895bis]

Table 1: Namespace Prefixes

3. Schema Mount

The schema mount mechanism defined in this document provides a new extensibility mechanism for use with YANG 1.1. In contrast to the existing mechanisms described in Section 1, schema mount defines the relationship between the source and target YANG modules outside these modules. The procedure consists of two separate steps that are described in the following subsections.

3.1. Mount Point Definition

A "container" or "list" node becomes a mount point if the "mount-point" extension (defined in the "ietf-yang-schema-mount" module) is used in its definition. This extension can appear only as a substatement of "container" and "list" statements.

The argument of the "mount-point" extension is a YANG identifier that defines a label for the mount point. A module MAY contain multiple "mount-point" statements having the same argument.

It is therefore up to the designer of the parent schema to decide about the placement of mount points. A mount point can also be made conditional by placing "if-feature" and/or "when" as substatements of the "container" or "list" statement that represents the mount point.

The "mount-point" statement MUST NOT be used in a YANG version 1 module [RFC6020]. The reason for this is that otherwise it is not possible to invoke mounted RPC operations, and receive mounted notifications. See Section 5 for details. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

Note that the "mount-point" statement does not define a new data node.

3.2. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
module: ietf-yang-schema-mount
  +--ro schema-mounts
     +--ro namespace* [prefix]
     |  +--ro prefix    yang:yang-identifier
     |  +--ro uri?      inet:uri
     +--ro mount-point* [module label]
        +--ro module                yang:yang-identifier
        +--ro label                 yang:yang-identifier
        +--ro config?               boolean
        +--ro (schema-ref)
           +--:(inline)
           |  +--ro inline!
           +--:(shared-schema)
              +--ro shared-schema!
                 +--ro parent-reference*   yang:xpath1.0
```

3.3. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are determined from state data in the "/schema-mounts" container.

Generally, the modules that are mounted under a mount point have no relation to the modules in the parent schema; specifically, if a module is mounted it may or may not be present in the parent schema and, if present, its data will generally have no relationship to the data of the parent. Exceptions are possible and such needs to be defined in the model defining the exception. For example, [I-D.ietf-rtgwg-lne-model] defines a mechanism to bind interfaces to mounted logical network elements.

The "/schema-mounts" container has the "mount-point" list as one of its children. Every entry of this list refers through its key to a mount point and specifies the mounted schema.

If a mount point is defined in the parent schema but does not have an entry in the "mount-point" list, then the mounted schema is void, i.e., instances of that mount point MUST NOT contain any data except those that are defined in the parent schema.

If multiple mount points with the same name are defined in the same module - either directly or because the mount point is defined in a grouping and the grouping is used multiple times - then the

corresponding "mount-point" entry applies equally to all such mount points.

The "config" property of mounted schema nodes is overridden and all nodes in the mounted schema are read-only ("config false") if at least one of the following conditions is satisfied for a mount point:

- the mount point is itself defined as "config false"
- the "config" leaf in the corresponding entry of the "mount-point" list is set to "false".

An entry of the "mount-point" list can specify the mounted schema in two different ways, "inline" or "shared-schema".

The mounted schema is determined at run time: every instance of the mount point that exists in the operational state MUST contain a copy of YANG library data that defines the mounted schema exactly as for a top-level schema. A client is expected to retrieve this data from the instance tree. In the "inline" case, instances of the same mount point MAY use different mounted schemas, whereas in the "shared-schema" case, all instances MUST use the same mounted schema. This means that in the "shared-schema" case, all instances of the same mount point MUST have the same YANG library content identifier. In the "inline" case, if two instances have the same YANG library content identifier it is not guaranteed that the YANG library contents are equal for these instances.

Examples of "inline" and "shared-schema" can be found in Appendix A.2 and Appendix A.3, respectively.

3.4. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which other schemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of mounted schemas. A schema for a mount point contained in a mounted module can be specified by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the schemas exactly as it is done in the top-level schema.

4. Referring to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted schema works exactly as a top-level schema, i.e., it is confined to the "mount jail". This means that all paths in the mounted schema (in leafrefs, instance-identifiers, XPath [XPATH] expressions, and target nodes of augments) are interpreted with the mount point as the

root node. YANG modules of the mounted schema as well as corresponding instance data thus cannot refer to schema nodes or instance data outside the mount jail.

However, this restriction is sometimes too severe. A typical example is network instances (NI) [I-D.ietf-rtgwg-ni-model], where each NI has its own routing engine but the list of interfaces is global and shared by all NIs. If we want to model this organization with the NI schema mounted using schema mount, the overall schema tree would look schematically as follows:

```
  +--rw interfaces
  |  +--rw interface* [name]
  |     ...
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name
        +--rw root
           +--rw routing
              ...
```

Here, the "root" node is the mount point for the NI schema. Routing configuration inside an NI often needs to refer to interfaces (at least those that are assigned to the NI), which is impossible unless such a reference can point to a node in the parent schema (interface name).

Therefore, schema mount also allows for such references. For every mount point in the "shared-schema" case, it is possible to specify a leaf-list named "parent-reference" that contains zero or more XPath 1.0 expressions. Each expression is evaluated with the node in the parent data tree where the mount point is defined as the context node. The result of this evaluation MUST be a nodeset (see the description of the "parent-reference" node for a complete definition of the evaluation context). For the purposes of evaluating XPath expressions within the mounted data tree, the union of all such nodesets is added to the accessible data tree.

It is worth emphasizing that the nodes specified in "parent-reference" leaf-list are available in the mounted schema only for XPath evaluations. In particular, they cannot be accessed there via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation as if it were defined as an action for the corresponding mount point, see Section 7.15 of [RFC7950]. An example of this is given in Appendix A.4.

Similarly, if the server emits a notification defined at the top level of any mounted module, it MUST be represented as if the notification was connected to the mount point, see Section 7.16 of [RFC7950].

Note, inline actions and notifications will not work when they are contained within a list node without a "key" statement (see section 7.15 and 7.16 of [RFC7950]). Therefore, to be useful, mount points that contain modules with RPCs, actions, and notifications SHOULD NOT have any ancestor node that is a list node without a "key" statement. This requirement applies to the definition of modules using the "mount-point" extension statement.

6. Network Management Datastore Architecture (NMDA) Considerations

The schema mount solution presented in this document is designed to work both with servers that implement the NMDA [RFC8342], and old servers that don’t implement the NMDA.

Note to RFC Editor: please update the date YYYY-MM-DD below with the revision of the ietf-yang-library in the published version of draft-ietf-netconf-rfc7895bis, and remove this note.

Specifically, a server that doesn’t support the NMDA, MAY implement revision 2016-06-21 of "ietf-yang-library" [RFC7895] under a mount point. A server that supports the NMDA, MUST implement at least revision YYYY-MM-DD of "ietf-yang-library" [I-D.ietf-netconf-rfc7895bis] under the mount points.

7. Interaction with the Network Configuration Access Control Model (NACM)

If NACM [RFC8341] is implemented on a server, it is used to control access to nodes defined by the mounted schema in the same way as for nodes defined by the top-level schema.

For example, suppose the module "ietf-interfaces" is mounted in the "root" container in the "logical-network-element" list defined in [I-D.ietf-rtgwg-lne-model]. Then the following NACM path can be used to control access to the "interfaces" container (where the character

'\' is used where a line break has been inserted for formatting reasons):

```
  <path xmlns:lne=
          "urn:ietf:params:xml:ns:yang:ietf-logical-network-element"
        xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    /lne:logical-network-elements\
      /lne:logical-network-element/lne:root/if:interfaces
  </path>
```

8. Implementation Notes

Network management of devices that use a data model with schema mount can be implemented in different ways. However, the following implementations options are envisioned as typical:

- shared management: instance data of both parent and mounted schemas are accessible within the same management session.

- split management: one (master) management session has access to instance data of both parent and mounted schemas but, in addition, an extra session exists for every instance of the mount point, having access only to the mounted data tree.

9. Schema Mount YANG Module

This module references [RFC6991].

<CODE BEGINS> file "ietf-yang-schema-mount@2018-10-16"

```
module ietf-yang-schema-mount {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
  prefix yangmnt;

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

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

  organization
```

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

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

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

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

    // 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 operational state data that specify the
       overall structure of the data model.

       Copyright (c) 2018 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', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in the module text are to be interpreted
       as described in BCP 14 [RFC 2119] [RFC8174] when, and only when,
       they appear in all capitals, as shown here.

       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.";

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2018-10-16 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Schema Mount";
    }
```

```
     /*
      * Extensions
      */

     extension mount-point {
       argument label;
       description
         "The argument 'label' is a YANG identifier, i.e., it is of the
          type 'yang:yang-identifier'.

          The 'mount-point' statement MUST NOT be used in a YANG
          version 1 module, neither explicitly nor via a 'uses'
          statement.

          The 'mount-point' statement MAY be present as a substatement
          of 'container' and 'list', and MUST NOT be present elsewhere.
          There MUST NOT be more than one 'mount-point' statement in a
          given 'container' or 'list' statement.

          If a mount point is defined within a grouping, its label is
          bound to the module where the grouping is used.

          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
          /schema-mounts/mount-point list with detailed information on
          which data models are mounted at each mount point.

          Note that the 'mount-point' statement does not define a new
          data node.";
     }

     /*
      * State data nodes
      */

     container schema-mounts {
       config false;
       description
         "Contains information about the structure of the overall
          mounted data model implemented in the server.";
       list namespace {
         key "prefix";
         description
           "This list provides a mapping of namespace prefixes that are
            used in XPath expressions of 'parent-reference' leafs to the
            corresponding namespace URI references.";
         leaf prefix {
```

```
          type yang:yang-identifier;
          description
            "Namespace prefix.";
        }
        leaf uri {
          type inet:uri;
          description
            "Namespace URI reference.";
        }
      }
      list mount-point {
        key "module label";
        description
          "Each entry of this list specifies a schema for a particular
           mount point.

           Each mount point MUST be defined using the 'mount-point'
           extension in one of the modules listed in the server's
           YANG library instance with conformance type 'implement'.";
        leaf module {
          type yang:yang-identifier;
          description
            "Name of a module containing the mount point.";
        }
        leaf label {
          type yang:yang-identifier;
          description
            "Label of the mount point defined using the 'mount-point'
             extension.";
        }
        leaf config {
          type boolean;
          default "true";
          description
            "If this leaf is set to 'false', then all data nodes in the
             mounted schema are read-only (config false), regardless of
             their 'config' property.";
        }
        choice schema-ref {
          mandatory true;
          description
            "Alternatives for specifying the schema.";
          container inline {
            presence
              "A complete self-contained schema is mounted at the
               mount point.";
            description
              "This node indicates that the server has mounted at least
```

```
            the module 'ietf-yang-library' at the mount point, and
            its instantiation provides the information about the
            mounted schema.

            Different instances of the mount point may have
            different schemas mounted.";
        }
        container shared-schema {
          presence
            "The mounted schema together with the 'parent-reference'
             make up the schema for this mount point.";
          description
            "This node indicates that the server has mounted at least
             the module 'ietf-yang-library' at the mount point, and
             its instantiation provides the information about the
             mounted schema.  When XPath expressions in the mounted
             schema are evaluated, the 'parent-reference' leaf-list
             is taken into account.

             Different instances of the mount point MUST have the
             same schema mounted.";
          leaf-list parent-reference {
            type yang:xpath1.0;
            description
              "Entries of this leaf-list are XPath 1.0 expressions
               that are evaluated in the following context:

                - The context node is the node in the parent data tree
                  where the mount-point is defined.

                - The accessible tree is the parent data tree
                  *without* any nodes defined in modules that are
                  mounted inside the parent schema.

                - The context position and context size are both equal
                  to 1.

                - The set of variable bindings is empty.

                - The function library is the core function library
                  defined in [XPath] and the functions defined in
                  Section 10 of [RFC7950].

                - The set of namespace declarations is defined by the
                  'namespace' list under 'schema-mounts'.

               Each XPath expression MUST evaluate to a nodeset
               (possibly empty). For the purposes of evaluating XPath
```

```
                    expressions whose context nodes are defined in the
                    mounted schema, the union of all these nodesets
                    together with ancestor nodes are added to the
                    accessible data tree.

                    Note that in the case 'ietf-yang-schema-mount' is
                    itself mounted, a 'parent-reference' in the mounted
                    module may refer to nodes that were brought into the
                    accessible tree through a 'parent-reference' in the
                    parent schema.";
              }
            }
          }
        }
      }
    }
```

<CODE ENDS>

10. 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
```

11. Security Considerations

YANG module "ietf-yang-schema-mount" 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 network configuration access control model [RFC8341] 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.

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:

- /schema-mounts: The schema defined by this state data provides detailed information about a server implementation may help an attacker identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules included in the schema, module revisions, module features, or even module deviations. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the schema information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

It is important to take the security considerations for all nodes in the mounted schemas into account, and control access to these nodes by using the mechanism described in Section 7.

Care must be taken when the "parent-reference" XPath expressions are constructed, since the result of the evaluation of these expressions is added to the accessible tree for any XPath expression found in the mounted schema.

12. 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]); and Lou Berger and Christian Hopps:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Alexander Clemm, Huawei, <alexander.clemm@huawei.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Jan Medved, Cisco, <jmedved@cisco.com>
- Eric Voit, Cisco, <evoit@cisco.com>

13. References

13.1. Normative References

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[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>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://www.rfc-editor.org/info/rfc7895>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

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

13.2. Informative References

[I-D.clemm-netmod-mount] Clemm, A., Voit, E., and J. Medved, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-06 (work in progress), March 2017.

[I-D.ietf-isis-yang-isis-cfg] Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-24 (work in progress), August 2018.

[I-D.ietf-rtgwg-device-model] Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

[I-D.ietf-rtgwg-lne-model] Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Logical Network Elements", draft-ietf-rtgwg-lne-model-10 (work in progress), March 2018.

[I-D.ietf-rtgwg-ni-model] Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Network Instances", draft-ietf-rtgwg-ni-model-12 (work in progress), March 2018.

[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, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

Appendix A. Example: Device Model with LNEs and NIs

This non-normative example demonstrates an implementation of the device model as specified in Section 2 of [I-D.ietf-rtgwg-device-model], using both logical network elements (LNE) and network instances (NI).

In these examples, the character '\' is used where a line break has been inserted for formatting reasons.

A.1. Physical Device

The data model for the physical device may be described by this YANG library content, assuming the server supports the NMDA:

```
{
   "ietf-yang-library:yang-library": {
     "content-id": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
     "module-set": [
       {
         "name": "physical-device-modules",
         "module": [
           {
             "name": "ietf-datastores",
             "revision": "2018-02-14",
             "namespace":
               "urn:ietf:params:xml:ns:yang:ietf-datastores"
           },
           {
             "name": "iana-if-type",
             "revision": "2015-06-12",
             "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type"
           },
           {
             "name": "ietf-interfaces",
             "revision": "2018-02-20",
```

```
              "feature": ["arbitrary-names", "pre-provisioning" ],
              "namespace":
                "urn:ietf:params:xml:ns:yang:ietf-interfaces"
            },
            {
              "name": "ietf-ip",
              "revision": "2018-02-22",
              "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip"
            },
            {
              "name": "ietf-logical-network-element",
              "revision": "2016-10-21",
              "feature": [ "bind-lne-name" ],
              "namespace":
                "urn:ietf:params:xml:ns:yang:\
                ietf-logical-network-element"
            },
            {
              "name": "ietf-yang-library",
              "revision": "2018-02-21",
              "namespace":
                "urn:ietf:params:xml:ns:yang:ietf-yang-library"
            },
            {
              "name": "ietf-yang-schema-mount",
              "revision": "2018-03-20",
              "namespace":
                "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount"
            }
          ],
          "import-only-module": [
            {
              "name": "ietf-inet-types",
              "revision": "2013-07-15",
              "namespace":
                "urn:ietf:params:xml:ns:yang:ietf-inet-types"
            },
            {
              "name": "ietf-yang-types",
              "revision": "2013-07-15",
              "namespace":
                "urn:ietf:params:xml:ns:yang:ietf-yang-types"
            }
          ]
        }
      ],
      "schema": [
        {
```

```
            "name": "physical-device-schema",
            "module-set": [ "physical-device-modules" ]
          }
        ],
        "datastore": [
          {
            "name": "ietf-datastores:running",
            "schema": "physical-device-schema"
          },
          {
            "name": "ietf-datastores:operational",
            "schema": "physical-device-schema"
          }
        ]
      }
    }
```

A.2. Logical Network Elements

Each LNE can have a specific data model that is determined at run time, so it is appropriate to mount it using the "inline" method, hence the following "schema-mounts" data:

```
{
  "ietf-yang-schema-mount:schema-mounts": {
    "mount-point": [
      {
        "module": "ietf-logical-network-element",
        "label": "root",
        "inline": {}
      }
    ]
  }
}
```

An administrator of the host device has to configure an entry for each LNE instance, for example,

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-logical-network-element:bind-lne-name": "eth0"
      }
    ]
  },
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "lne-1",
        "managed": true,
        "description": "LNE with NIs",
        "root": {
          ...
        }
      }
      ...
    ]
  }
}
```

and then also place necessary state data as the contents of the "root" instance, which should include at least

- YANG library data specifying the LNE's data model, for example, assuming the server does not implement the NMDA:

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "9358e11874068c8be06562089e94a89e0a392019",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-inet-types",
        "revision": "2013-07-15",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
        "conformance-type": "import"
      },
```

```
        {
          "name": "ietf-interfaces",
          "revision": "2014-05-08",
          "feature": [
            "arbitrary-names",
            "pre-provisioning"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "feature": [
            "ipv6-privacy-autoconf"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-network-instance",
          "revision": "2016-10-27",
          "feature": [
            "bind-network-instance-name"
          ],
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-network-instance",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-schema-mount",
          "revision": "2017-05-16",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-types",
          "revision": "2013-07-15",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "conformance-type": "import"
        }
```

```
      ]
    }
  }
```

o state data for interfaces assigned to the LNE instance (that effectively become system-controlled interfaces for the LNE), for example:

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "statistics": {
          "discontinuity-time": "2016-12-16T17:11:27+02:00"
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "fe80::42a8:f0ff:fea8:24fe",
              "origin": "link-layer",
              "prefix-length": 64
            }
          ]
        }
      }
    ]
  }
}
```

A.3. Network Instances

Assuming that network instances share the same data model, it can be mounted using the "shared-schema" method as follows:

```
{
  "ietf-yang-schema-mount:schema-mounts": {
    "namespace": [
      {
          "prefix": "if",
          "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
      },
      {
          "prefix": "ni",
          "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
      }
    ],
    "mount-point": [
      {
        "module": "ietf-network-instance",
        "label": "root",
          "shared-schema": {
            "parent-reference": [
              "/if:interfaces/if:interface[\
              ni:bind-network-instance-name = current()/../ni:name]"
            ]
          }
      }
    ]
  }
}
```

Note also that the "ietf-interfaces" module appears in the "parent-reference" leaf-list for the mounted NI schema. This means that references to LNE interfaces, such as "outgoing-interface" in static routes, are valid despite the fact that "ietf-interfaces" isn't part of the NI schema.

A.4. Invoking an RPC Operation

Assume that the mounted NI data model also implements the "ietf-isis" module [I-D.ietf-isis-yang-isis-cfg]. An RPC operation defined in this module, such as "clear-adjacency", can be invoked by a client session of a LNE's RESTCONF server as an action tied to a the mount point of a particular network instance using a request URI like this (all on one line):

```
POST /restconf/data/ietf-network-instance:network-instances/
    network-instance=rtrA/root/ietf-isis:clear-adjacency HTTP/1.1
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
March 13, 2017

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-01

Abstract

This document defines YANG modules to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. These modules allow configuration of Layer 3 and Layer 2 sub-interfaces (e.g. attachment circuits) that can interoperate with IETF based forwarding protocols; such as IP and L3VPN services; or L2VPN services like VPWS, VPLS, and EVPN. The sub-interfaces also interoperate with VLAN tagged traffic orginating from an IEEE 802.1Q compliant bridge. Primarily the classification is based on VLAN identifiers in the 802.1Q VLAN tags, but the model also has support for matching on some other layer 2 frame header fields and is designed to be extensible to match on other arbitrary header fields.

The model differs from an IEEE 802.1Q bridge model in that the configuration is interface/sub-interface based as opposed to being based on membership of an 802.1Q VLAN bridge.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
1.1. Terminology 3
1.2. Tree Diagrams 4
2. Objectives 4
2.1. Interoperability with IEEE 802.1Q compliant bridges . . . 4
2.2. Extensibility 4
3. L3 Interface VLAN Model 5
4. Flexible Encapsulation Model 5
5. L3 Interface VLAN YANG Module 7
6. Flexible Encapsulation YANG Module 10
7. Acknowledgements 19
8. ChangeLog 19
8.1. WG version -01 19
8.2. Version -04 20
8.3. Version -03 20
9. IANA Considerations 20
10. Security Considerations 20
10.1. if-l3-vlan.yang 21
10.2. flexible-encapsulation.yang 21
11. References 23
11.1. Normative References 23
11.2. Informative References 23
Appendix A. Comparison with the IEEE 802.1Q Configuration Model 24
A.1. Sub-interface based configuration model overview 24
A.2. IEEE 802.1Q Bridge Configuration Model Overview 25
A.3. Possible Overlap Between the Two Models 26
Authors' Addresses 26

1. Introduction

This document defines two YANG [RFC7950] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang] and the generic interfaces data model defined in [RFC7223]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature and service configuration applied to them.

The purpose of these models is to allow IETF defined forwarding protocols, such as IPv6 [RFC2460], Ethernet Pseudo Wires [RFC4448] and VPLS [RFC4761] [RFC4762] to be configurable via YANG when interoperating with VLAN tagged traffic received from an IEEE 802.1Q compliant bridge.

In the case of layer 2 Ethernet services, the flexible encapsulation module also supports flexible rewriting of the VLAN tags contained the in frame header.

For reference, a comparison between the sub-interface based YANG model documented in this draft and an IEEE 802.1Q bridge model is described in Appendix A.

In summary, the YANG modules defined in this internet draft are:

if-l3-vlan.yang - Defines the model for basic classification of VLAN tagged traffic to L3 transport services

flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic to L2 transport services

1.1. Terminology

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

Sub-interface: A sub-interface is a small augmentation of a regular interface in the standard YANG module for Interface Management that represents a subset of the traffic handled by its parent interface. As such, it supports both configuration and operational data using any other YANG models that augment or reference interfaces in the normal way. It is defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

1.2. Tree Diagrams

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

- 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 primary aim of the YANG modules contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices that is fully compatible with IEEE 802.1Q complaint bridges.

A secondary aim is for the modules to be structured in such a way that they can be cleanly extended in future.

2.1. Interoperability with IEEE 802.1Q compliant bridges

The modules defined in this document are designed to fully interoperate with IEEE 802.1Q compliant bridges. In particular, the models are restricted to only matching, adding, or rewriting the 802.1Q VLAN tags in frames in ways that are compatible with IEEE 802.1Q compliant bridges.

2.2. Extensibility

The modules are structured in such a way that they can be sensibly extended. In particular:

The tag stack is represented as a list to allow a tag stack of more than two tags to be supported if necessary in future.

The tag stack list elements allow other models, or vendors, to include additional forms of tag matching and rewriting. The

intention, however, is that it should not be necessary to have any vendor specific extensions to any of the YANG models defined in this document to implement standard Ethernet and VLAN services.

3. L3 Interface VLAN Model

The L3 Interface VLAN model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface based L3 service. It allows for termination of traffic with up to two 802.1Q VLAN tags.

The "if-l3-vlan" YANG module has the following structure:

```
module: ietf-if-l3-vlan
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
                                                  if-cmn:encaps-type:
    +--:(vlan)
       +--rw vlan
          +--rw tag* [index]
             +--rw index        uint8
             +--rw dot1q-tag
                +--rw tag-type    dot1q-tag-type
                +--rw vlan-id     ieee:vlanid
```

4. Flexible Encapsulation Model

The Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 headers before the frame is handed off to the appropriate forwarding code for further handling.

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that manipulations are generally implemented in a symmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress

traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The structure of the model is currently limited to matching or rewriting a maximum of two 802.1Q tags in the frame header but has been designed to be easily extensible to matching/rewriting three or more VLAN tags in future, if required.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

The "flexible-encapsulation" YANG module has the following structure:

```
module: ietf-flexible-encapsulation
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
                                                  if-cmn:encaps-type:
    +--:(flexible) {flexible-encapsulation-rewrites}?
       +--rw flexible
          +--rw match
          |  +--rw (match-type)
          |     +--:(default)
          |     |  +--rw default?           empty
          |     +--:(untagged)
          |     |  +--rw untagged?          empty
          |     +--:(priority-tagged)
          |     |  +--rw priority-tagged
          |     |     +--rw tag-type?   dot1q-types:dot1q-tag-type
          |     +--:(vlan-tagged)
          |        +--rw vlan-tagged
          |           +--rw tag* [index]
          |           |  +--rw index        uint8
          |           |  +--rw dot1q-tag
          |           |     +--rw tag-type    dot1q-tag-type
          |           |     +--rw vlan-id     union
          |           +--rw match-exact-tags?   empty
          +--rw rewrite {flexible-rewrites}?
             +--rw (direction)?
                +--:(symmetrical)
                |  +--rw symmetrical
                |     +--rw tag-rewrite {tag-rewrites}?
                |        +--rw pop-tags?   uint8
                |        +--rw push-tag* [index]
```

```
                  |           +--rw index        uint8
                  |           +--rw dot1q-tag
                  |              +--rw tag-type    dot1q-tag-type
                  |              +--rw vlan-id     ieee:vlanid
                  +--:(asymmetrical) {asymmetric-rewrites}?
                     +--rw ingress
                     |  +--rw tag-rewrite {tag-rewrites}?
                     |     +--rw pop-tags?   uint8
                     |     +--rw push-tag* [index]
                     |        +--rw index        uint8
                     |        +--rw dot1q-tag
                     |           +--rw tag-type    dot1q-tag-type
                     |           +--rw vlan-id     ieee:vlanid
                     +--rw egress
                        +--rw tag-rewrite {tag-rewrites}?
                           +--rw pop-tags?   uint8
                           +--rw push-tag* [index]
                              +--rw index        uint8
                              +--rw dot1q-tag
                                 +--rw tag-type    dot1q-tag-type
                                 +--rw vlan-id     ieee:vlanid
  augment /if:interfaces/if:interface:
    +--rw flexible-encapsulation
       +--rw local-traffic-default-encaps
          +--rw tag* [index]
             +--rw index        uint8
             +--rw dot1q-tag
                +--rw tag-type    dot1q-tag-type
                +--rw vlan-id     ieee:vlanid
```

5. L3 Interface VLAN YANG Module

This YANG module augments the encapsultion container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

```
<CODE BEGINS> file "ietf-if-l3-vlan@2017-03-13.yang"
module ietf-if-l3-vlan {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-if-l3-vlan";

  prefix if-l3-vlan;

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

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

     import ieee802-dot1q-types {
       prefix dot1q-types;
     }

     import ietf-interfaces-common {
       prefix if-cmn;
     }

     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 YANG module models L3 VLAN sub-interfaces";

     revision 2017-03-13 {
       description "Latest draft revision";

       reference
         "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-01";
     }

     /*
      * Add support for the 802.1Q VLAN encapsulation syntax on layer 3
      * terminated VLAN sub-interfaces.
      */
     augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
             "if-cmn:encaps-type" {
       when "../if:type = 'ianaift:l2vlan' and
             derived-from-or-self(../if-cmn:forwarding-mode,
                                  'if-cmn:network-layer')" {
         description
```

```
          "Applies only to VLAN sub-interfaces that are operating at
           layer 3";
        }
        description
          "Augment the generic interface encapsulation with an
           encapsulation for layer 3 VLAN sub-interfaces";

        /*
         * Matches a VLAN, or pair of VLAN Ids to classify traffic
         * into an L3 service.
         */
        case vlan {
          container vlan {
            description
              "Match VLAN tagged frames with specific VLAN Ids";
            list tag {
              must 'index != 0 or ' +
                   'count(../tag/index) != 2 or ' +
                   'dot1q-tag/tag-type = "s-vlan"' {
                error-message
                  "When matching two tags, the outer tag must be of
                   S-VLAN tag type";
                description
                  "For IEEE 802.1Q interoperability, when matching two
                   tags, it is required that the outer tag is an S-VLAN,
                   and the inner tag is a C-VLAN";
              }

              must 'index != 1 or ' +
                   'count(../tag/index) != 2 or ' +
                   'dot1q-tag/tag-type = "c-vlan"' {
                error-message
                  "When matching two tags, the inner tag must be of
                   C-VLAN tag type";
                description
                  "For IEEE 802.1Q interoperability, when matching two
                   tags, it is required that the outer tag is an S-VLAN,
                   and the inner tag is a C-VLAN";
              }

              key "index";
              min-elements 1;
              max-elements 2;

              description
                "The tags to match, with the outermost tag to match with
                 index 0";
              leaf index {
```

```
              type uint8 {
                range "0..1";
              }

              /*
               * Only allow matching on an inner tag (at index 1), if
               * also matching on the outer tag at the same time.
               */
              must ". = 0 or
                    count(../../tag[index = 0]/index) > 0" {
                error-message
                  "An inner tag can only be matched on when also
                   matching on an outer tag";
                description
                  "Only allow matching on an inner tag, if also
                   matching on the outer tag at the same time";
              }

              description
                "The index into the tag stack, outermost tag first";
            }

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Flexible Encapsulation YANG Module

This YANG module augments the encapsultion container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

This YANG module also augments the interface container defined in [RFC7223].

```
<CODE BEGINS> file "ietf-flexible-encapsulation@2017-03-13.yang"
module ietf-flexible-encapsulation {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-flexible-encapsulation";

  prefix flex;
```

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

     import ietf-interfaces-common {
       prefix if-cmn;
     }

     import ieee802-dot1q-types {
       prefix dot1q-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>

        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 YANG module describes interface configuration for flexible
        VLAN matches and rewrites.";

     revision 2017-03-13 {
       description "Latest draft revision";

       reference
         "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-01";
     }

     feature flexible-encapsulation-rewrites {
       description
         "This feature indicates whether the network element supports
          flexible Ethernet encapsulation that allows for matching VLAN
          ranges and performing independent tag manipulations";
     }

     feature flexible-rewrites {
       description
```

```
        "This feature indicates whether the network element supports
          specifying flexible rewrite operations";
      }

      feature asymmetric-rewrites {
        description
          "This feature indicates whether the network element supports
           specifying different rewrite operations for the ingress
           rewrite operation and egress rewrite operation.";
      }

      feature tag-rewrites {
        description
          "This feature indicates whether the network element supports
           the flexible rewrite functionality specifying flexible tag
           rewrites";
      }

      /*
       * flexible-match grouping.
       *
       * This grouping represents a flexible match.
       *
       * The rules for a flexible match are:
       *     1. default, untagged, priority tag, or a stack of tags.
       *   - Each tag in the stack of tags matches:
       *      1. tag type (802.1Q or 802.1ad) +
       *      2. tag value:
       *        i. single tag
       *        ii. set of tag ranges/values.
       *        iii. "any" keyword
       */
      grouping flexible-match {
        description "Flexible match";
        choice match-type {
          mandatory true;
          description "Provides a choice of how the frames may be
                       matched";

          case default {
            description "Default match";
            leaf default {
              type empty;
              description
                "Default match.  Matches all traffic not matched to any
                 other peer sub-interface by a more specific
                 encapsulation.";
            } // leaf default
```

```
        } // case default

        case untagged {
          description "Match untagged Ethernet frames only";
          leaf untagged {
            type empty;
            description
              "Untagged match.  Matches all untagged traffic.";
          } // leaf untagged
        } // case untagged

        case priority-tagged {
          description "Match priority tagged Ethernet frames only";

          container priority-tagged {
            description "Priority tag match";
            leaf tag-type {
              type dot1q-types:dot1q-tag-type;
              description "The 802.1Q tag type of matched priority
                           tagged packets";
            }
          }
        }

        case vlan-tagged {
          container vlan-tagged {
            description "Matches VLAN tagged frames";
            list tag {
              must 'index != 0 or ' +
                   'count(../tag/index) != 2 or ' +
                   'dot1q-tag/tag-type = "s-vlan"' {
                error-message
                  "When matching two tags, the outer tag must be of
                   S-VLAN tag type";
                description
                  "For IEEE 802.1Q interoperability, when matching two
                   tags, it is required that the outer tag is an
                   S-VLAN, and the inner tag is a C-VLAN";
              }

              must 'index != 1 or ' +
                   'count(../tag/index) != 2 or ' +
                   'dot1q-tag/tag-type = "c-vlan"' {
                error-message
                  "When matching two tags, the inner tag must be of
                   C-VLAN tag type";
                description
                  "For IEEE 802.1Q interoperability, when matching two
```

```
                    tags, it is required that the outer tag is an
                    S-VLAN, and the inner tag is a C-VLAN";
                }

                key "index";
                min-elements 1;
                max-elements 2;
                description "The tags to match, with the outermost tag to
                             match assigned index 0";
                leaf index {
                  type uint8 {
                    range "0..1";
                  }

                  must ". = 0 or
                        count(../../tag[index = 0]/index) > 0" {
                    error-message "An inner tag can only be matched on
                                   when also matching on an outer tag";
                    description "Only allow matching on an inner tag, if
                                 also matching on the outer tags at the
                                 same time";
                  }
                  description
                    "The index into the tag stack, outermost tag first";
                }

                uses
                  'dot1q-types:'+
                  'dot1q-tag-ranges-or-any-classifier-grouping';
              }

              leaf match-exact-tags {
                type empty;
                description
                  "If set, indicates that all 802.1Q VLAN tags in the
                   Ethernet frame header must be explicitly matched, i.e.
                   the EtherType following the matched tags must not be a
                   802.1Q tag EtherType.  If unset then extra 802.1Q VLAN
                   tags are allowed.";
              }
            }
          }
        } // encaps-type
      }

      /*
       * Grouping for tag-rewrite that can be expressed either
       * symmetrically, or in the ingress and/or egress directions
```

```
       * independently.
       */
      grouping tag-rewrite {
        description "Flexible rewrite";
        leaf pop-tags {
          type uint8 {
            range 1..2;
          }
          description "The number of tags to pop (or translate if used in
                       conjunction with push-tags)";
        }

        list push-tag {
          must 'index != 0 or ' +
               'count(../push-tag/index) != 2 or ' +
               'dot1q-tag/tag-type = "s-vlan"' {
            error-message
              "When pushing two tags, the outer tag must be of
               S-VLAN tag type";
            description
              "For IEEE 802.1Q interoperability, when pushing two
               tags, it is required that the outer tag is an
               S-VLAN, and the inner tag is a C-VLAN";
          }

          must 'index != 1 or ' +
               'count(../push-tag/index) != 2 or ' +
               'dot1q-tag/tag-type = "c-vlan"' {
            error-message
              "When pushing two tags, the inner tag must be of
               C-VLAN tag type";
            description
              "For IEEE 802.1Q interoperability, when pushing two
               tags, it is required that the outer tag is an
               S-VLAN, and the inner tag is a C-VLAN";
          }

          key "index";
          max-elements 2;
          description "The number of tags to push (or translate if used
                       in conjunction with pop-tags)";
          /*
           * Server should order by increasing index.
           */
          leaf index {
            type uint8 {
              range 0..1;
            }
```

```
          /*
           * Only allow a push of an inner tag if an outer tag is also
           * being pushed.
           */
          must ". != 0 or
                count(../../push-tag[index = 0]/index) > 0" {
            error-message "An inner tag can only be pushed if an outer
                           tag is also specified";
            description "Only allow a push of an inner tag if an outer
                         tag is also being pushed";
          }

          description "The index into the tag stack";
        }

        uses dot1q-types:dot1q-tag-classifier-grouping;
      }
    }

    /*
     * Grouping for all flexible rewrites of fields in the L2 header.
     *
     * This currently only includes flexible tag rewrites, but is
     * designed to be extensible to cover rewrites of other fields in
     * the L2 header if required.
     */
    grouping flexible-rewrite {
      description "Flexible rewrite";

      /*
       * Tag rewrite.
       *
       * All tag rewrites are formed using a combination of pop-tags
       * and push-tags operations.
       */
      container tag-rewrite {
        if-feature tag-rewrites;
        description "Tag rewrite.  Translate operations are expressed
                     as a combination of tag push and pop operations.";
        uses tag-rewrite;
      }
    }
    augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
            "if-cmn:encaps-type" {
      when "../if:type = 'if-cmn:ethSubInterface' and
            derived-from-or-self(../if-cmn:forwarding-mode,
                                 'if-cmn:layer-2-forwarding')" {
        description "Applies only to Ethernet sub-interfaces that are
```

```
                operating at transport layer 2";
      }
      description
        "Add flexible match and rewrite for VLAN sub-interfaces";

      /*
       * A flexible encapsulation allows for the matching of ranges and
       * sets of VLAN Ids.  The structure is also designed to be
       * extended to allow for matching/rewriting other fields within
       * the L2 frame header if required.
       */
      case flexible {
        if-feature flexible-encapsulation-rewrites;
        description "Flexible encapsulation and rewrite";
        container flexible {
          description "Flexible encapsulation and rewrite";

          container match {
            description
              "The match used to classify frames to this interface";
            uses flexible-match;
          }

          container rewrite {
            if-feature flexible-rewrites;
            description "L2 frame rewrite operations";
            choice direction {
              description "Whether the rewrite policy is symmetrical or
                           asymmetrical";
              case symmetrical {
                container symmetrical {
                  uses flexible-rewrite;
                  description
                    "Symmetrical rewrite.  Expressed in the ingress
                     direction, but the reverse operation is applied
                     to egress traffic";
                }
              }

              /*
               * Allow asymmetrical rewrites to be specified.
               */
              case asymmetrical {
                if-feature asymmetric-rewrites;
                description "Asymmetrical rewrite";
                container ingress {
                  uses flexible-rewrite;
                  description "Ingress rewrite";
```

```
                }
                container egress {
                  uses flexible-rewrite;
                  description "Egress rewrite";
                }
              }
            }
          }
        }
      }

      augment "/if:interfaces/if:interface" {
        when "if:type = 'if-cmn:ethSubInterface' and
              derived-from-or-self(if-cmn:forwarding-mode,
                                   'if-cmn:layer-2-forwarding')" {
          description "Any L2 Ethernet sub-interfaces";
        }
        description "Add flexible encapsulation configuration for VLAN
                     sub-interfaces";

        /*
         * All flexible encapsulation specific interface configuration
         * (except for the actual encapsulation and rewrite) is contained
         * by a flexible-encapsulation container on the interface.
         */
        container flexible-encapsulation {
          description
            "All per interface flexible encapsulation related fields";

          /*
           * For encapsulations that match a range of VLANs (or Any),
           * allow configuration to specify the default VLAN tag values
           * to use for any traffic that is locally sourced from an
           * interface on the device.
           */
          container local-traffic-default-encaps {
            description "The VLAN tags to use by default for locally
                         sourced traffic";
            list tag {
              key "index";
              max-elements 2;

              description
                "The VLAN tags to use by locally sourced traffic";

              leaf index {
                type uint8 {
```

```
                range "0..1";
              }

              /*
               * Only allow an inner tag to be specified if an outer
               * tag has also been specified.
               */
              must ". = 0 or
                    count(../../tag[index = 0]/index) > 0" {
                error-message "An inner tag can only be specified if an
                               outer tag has also been specified";
                description "Ensure that an inner tag cannot be
                             specified without an outer tag'";
              }

              description "The index into the tag stack, outermost tag
                           assigned index 0";
            }

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }
        }
      }
    }
  }
  <CODE ENDS>
```

7. Acknowledgements

The authors would particularly like to thank John Messenger, Glenn Parsons, and Dan Romascanu for their help progressing this draft.

The authors would also like to thank Alex Campbell, Eric Gray, Giles Heron, Marc Holness, Iftekhar Hussain, Neil Ketley, William Lupton, John Messenger, Glenn Parsons, Ludwig Pauwels, and members of the IEEE 802.1 WG for their helpful reviews and feedback on this draft.

8. ChangeLog

8.1. WG version -01

- Tweaked the abstract.
- Removed unnecessary feature for the L3 sub-interface module.
- Update the 802.1Qcp type references.

- Remove extra tag container for L3 sub-interfaces YANG.

8.2. Version -04

- IEEE 802.1 specific types have been removed from the draft. These are now referenced from the 802.1Qcp draft YANG modules.

- Fixed errors in the xpath expressions.

8.3. Version -03

- Incorporates feedback received from presenting to the IEEE 802.1 WG.

- Updates the modules for double tag matches/rewrites to restrict the outer tag type to S-VLAN and inner tag type to C-VLAN.

- Updates the introduction to indicate primary use case is for IETF forwarding protocols.

- Updates the objectives to make IEEE 802.1Q bridge interoperability a key objective.

9. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

10. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

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

10.1. if-l3-vlan.yang

The nodes in the if-l3-vlan YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/vlan, that are sensitive to this are:

- tag
- tag/index
- tag/index/tag-type
- tag/index/vlan-id

10.2. flexible-encapsulation.yang

There are many nodes in the flexible-encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/match, that are sensitive to this are:

- default
- untagged
- priority-tagged
- priority-tagged/tag-type
- vlan-tagged
- vlan-tagged/index
- vlan-tagged/index/dot1q-tag/vlan-type
- vlan-tagged/index/dot1q-tag/vlan-id
- vlan-tagged/match-exact-tags

There are also many modes in the flexible-encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/rewrite, that are sensitive to this are:

- symmetrical/tag-rewrite/pop-tags
- symmetrical/tag-rewrite/push-tag
- symmetrical/tag-rewrite/push-tag/index
- symmetrical/tag-rewrite/push-tag/dot1q-tag/tag-type
- symmetrical/tag-rewrite/push-tag/dot1q-tag/vlan-id
- asymmetrical/ingress/tag-rewrite/pop-tags
- asymmetrical/ingress/tag-rewrite/push-tag
- asymmetrical/ingress/tag-rewrite/push-tag/index
- asymmetrical/ingress/tag-rewrite/push-tag/dot1q-tag/tag-type
- asymmetrical/ingress/tag-rewrite/push-tag/dot1q-tag/vlan-id
- asymmetrical/egress/tag-rewrite/pop-tags
- asymmetrical/egress/tag-rewrite/push-tag
- asymmetrical/egress/tag-rewrite/push-tag/index
- asymmetrical/egress/tag-rewrite/push-tag/dot1q-tag/tag-type
- asymmetrical/egress/tag-rewrite/push-tag/dot1q-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

- tag

- tag/index
- tag/dot1q-tag/tag-type
- tag/dot1q-tag/vlan-id

11. References

11.1. Normative References

[I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tsingh@juniper.net, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-04 (work in progress), March 2017.

[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>.

[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>.

[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>.

11.2. Informative References

[dot1Qcp] Holness, M., "802.1Qcp Bridges and Bridged Networks - Amendment: YANG Data Model", 2016.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <http://www.rfc-editor.org/info/rfc4448>.

[RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <http://www.rfc-editor.org/info/rfc4761>.

[RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <http://www.rfc-editor.org/info/rfc4762>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <http://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group is also developing a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.

- Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.

- Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.

- In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.

- The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.

- Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.

- Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.

- VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.

- Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

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

July 2, 2018

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-04

Abstract

This document defines YANG modules to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. These modules allow configuration of Layer 3 and Layer 2 sub-interfaces (e.g. attachment circuits) that can interoperate with IETF based forwarding protocols; such as IP and L3VPN services; or L2VPN services like VPWS, VPLS, and EVPN. The sub-interfaces also interoperate with VLAN tagged traffic orginating from an IEEE 802.1Q compliant bridge. Primarily the classification is based on VLAN identifiers in the 802.1Q VLAN tags, but the model also has support for matching on some other layer 2 frame header fields and is designed to be extensible to match on other arbitrary header fields.

The model differs from an IEEE 802.1Q bridge model in that the configuration is interface/sub-interface based as opposed to being based on membership of an 802.1Q VLAN bridge.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
1.1. Terminology 3
1.2. Tree Diagrams 4
2. Objectives 4
2.1. Interoperability with IEEE 802.1Q compliant bridges . . . 4
2.2. Extensibility 4
3. L3 Interface VLAN Model 5
4. Flexible Encapsulation Model 5
5. L3 Interface VLAN YANG Module 7
6. Flexible Encapsulation YANG Module 10
7. Examples 19
7.1. Layer 3 sub-interfaces with IPv6 19
7.2. Layer 2 sub-interfaces with L2VPN 21
8. Acknowledgements 23
9. ChangeLog 23
9.1. WG version -04 23
9.2. WG version -03 24
9.3. WG version -02 24
9.4. WG version -01 24
9.5. Version -04 24
9.6. Version -03 24
10. IANA Considerations 24
11. Security Considerations 25
11.1. if-l3-vlan.yang 25
11.2. flexible-encapsulation.yang 25
12. References 27
12.1. Normative References 27
12.2. Informative References 28
Appendix A. Comparison with the IEEE 802.1Q Configuration Model 29
A.1. Sub-interface based configuration model overview 29
A.2. IEEE 802.1Q Bridge Configuration Model Overview 30

A.3. Possible Overlap Between the Two Models 30
Authors' Addresses 31

1. Introduction

This document defines two YANG [RFC7950] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang] and the generic interfaces data model defined in [RFC8343]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature and service configuration applied to them.

The purpose of these models is to allow IETF defined forwarding protocols, such as IPv6 [RFC2460], Ethernet Pseudo Wires [RFC4448] and VPLS [RFC4761] [RFC4762] to be configurable via YANG when interoperating with VLAN tagged traffic received from an IEEE 802.1Q compliant bridge.

In the case of layer 2 Ethernet services, the flexible encapsulation module also supports flexible rewriting of the VLAN tags contained the in frame header.

For reference, a comparison between the sub-interface based YANG model documented in this draft and an IEEE 802.1Q bridge model is described in Appendix A.

In summary, the YANG modules defined in this internet draft are:

if-l3-vlan.yang - Defines the model for basic classification of VLAN tagged traffic to L3 transport services

flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic to L2 transport services

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Sub-interface: A sub-interface is a small augmentation of a regular interface in the standard YANG module for Interface Management that represents a subset of the traffic handled by its parent interface. As such, it supports both configuration and operational data using any other YANG models that augment or reference interfaces in the

normal way. It is defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

1.2. Tree Diagrams

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

- 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 primary aim of the YANG modules contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices that is fully compatible with IEEE 802.1Q compliant bridges.

A secondary aim is for the modules to be structured in such a way that they can be cleanly extended in future.

2.1. Interoperability with IEEE 802.1Q compliant bridges

The modules defined in this document are designed to fully interoperate with IEEE 802.1Q compliant bridges. In particular, the models are restricted to only matching, adding, or rewriting the 802.1Q VLAN tags in frames in ways that are compatible with IEEE 802.1Q compliant bridges.

2.2. Extensibility

The modules are structured in such a way that they can be sensibly extended. In particular:

The tag stack is represented as a list to allow a tag stack of more than two tags to be supported if necessary in future.

The tag stack list elements allow other models, or vendors, to include additional forms of tag matching and rewriting. The intention, however, is that it should not be necessary to have any vendor specific extensions to any of the YANG models defined in this document to implement standard Ethernet and VLAN services.

3. L3 Interface VLAN Model

The L3 Interface VLAN model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface based L3 service. It allows for termination of traffic with up to two 802.1Q VLAN tags.

The "if-l3-vlan" YANG module has the following structure:

```
module: ietf-if-l3-vlan
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
                                                  if-cmn:encaps-type:
    +--:(dot1q-vlan)
       +--rw dot1q-vlan
          +--rw outer-tag!
          |  +--rw tag-type    dot1q-tag-type
          |  +--rw vlan-id     ieee:vlanid
          +--rw second-tag!
             +--rw tag-type    dot1q-tag-type
             +--rw vlan-id     ieee:vlanid
```

4. Flexible Encapsulation Model

The Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 headers before the frame is handed off to the appropriate forwarding code for further handling.

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that

manipulations are generally implemented in a symmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The structure of the model is currently limited to matching or rewriting a maximum of two 802.1Q tags in the frame header but has been designed to be easily extensible to matching/rewriting three or more VLAN tags in future, if required.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

The "flexible-encapsulation" YANG module has the following structure:

```
module: ietf-flexible-encapsulation
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
                                                 if-cmn:encaps-type:
    +--:(flexible)
       +--rw flexible
          +--rw match
          |  +--rw (match-type)
          |     +--:(default)
          |     |  +--rw default?                empty
          |     +--:(untagged)
          |     |  +--rw untagged?               empty
          |     +--:(dot1q-priority-tagged)
          |     |  +--rw dot1q-priority-tagged
          |     |     +--rw tag-type?   dot1q-types:dot1q-tag-type
          |     +--:(dot1q-vlan-tagged)
          |        +--rw dot1q-vlan-tagged
          |           +--rw outer-tag!
          |           |  +--rw tag-type    dot1q-tag-type
          |           |  +--rw vlan-id     union
          |           +--rw second-tag!
          |           |  +--rw tag-type    dot1q-tag-type
          |           |  +--rw vlan-id     union
          |           +--rw match-exact-tags?   empty
          +--rw rewrite {flexible-rewrites}?
          |  +--rw (direction)?
          |     +--:(symmetrical)
```

```
|     |  +--rw symmetrical
|     |     +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|     |        +--rw pop-tags?    uint8
|     |        +--rw push-tags
|     |           +--rw outer-tag!
|     |           |  +--rw tag-type    dot1q-tag-type
|     |           |  +--rw vlan-id     ieee:vlanid
|     |           +--rw second-tag!
|     |              +--rw tag-type    dot1q-tag-type
|     |              +--rw vlan-id     ieee:vlanid
|     +--:(asymmetrical) {asymmetric-rewrites}?
|        +--rw ingress
|        |  +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|        |     +--rw pop-tags?    uint8
|        |     +--rw push-tags
|        |        +--rw outer-tag!
|        |        |  +--rw tag-type    dot1q-tag-type
|        |        |  +--rw vlan-id     ieee:vlanid
|        |        +--rw second-tag!
|        |           +--rw tag-type    dot1q-tag-type
|        |           +--rw vlan-id     ieee:vlanid
|        +--rw egress
|           +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|              +--rw pop-tags?    uint8
|              +--rw push-tags
|                 +--rw outer-tag!
|                 |  +--rw tag-type    dot1q-tag-type
|                 |  +--rw vlan-id     ieee:vlanid
|                 +--rw second-tag!
|                    +--rw tag-type    dot1q-tag-type
|                    +--rw vlan-id     ieee:vlanid
|
+--rw local-traffic-default-encaps!
   +--rw outer-tag!
   |  +--rw tag-type    dot1q-tag-type
   |  +--rw vlan-id     ieee:vlanid
   +--rw second-tag!
      +--rw tag-type    dot1q-tag-type
      +--rw vlan-id     ieee:vlanid
```

5. L3 Interface VLAN YANG Module

This YANG module augments the encapsultion container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

```
<CODE BEGINS> file "ietf-if-l3-vlan@2017-10-30.yang"
module ietf-if-l3-vlan {
```

```
yang-version 1.1;

namespace "urn:ietf:params:xml:ns:yang:ietf-if-l3-vlan";

prefix if-l3-vlan;

import ietf-interfaces {
  prefix if;
}

import iana-if-type {
  prefix ianaift;
}

import ieee802-dot1q-types {
  prefix dot1q-types;
}

import ietf-interfaces-common {
  prefix if-cmn;
}

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 YANG module models L3 VLAN sub-interfaces";

revision 2017-10-30 {
  description "Latest draft revision";

  reference
    "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-03";
}
```

```
     /*
      * Add support for the 802.1Q VLAN encapsulation syntax on layer 3
      * terminated VLAN sub-interfaces.
      */
     augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
             "if-cmn:encaps-type" {
       when
           "derived-from-or-self(../if:type,
                                 'ianaift:ethernetCsmacd') or
            derived-from-or-self(../if:type,
                                 'ianaift:ieee8023adLag') or
            derived-from-or-self(../if:type,
                                 'if-cmn:ethSubInterface')" {
         description
           "Applies only to Ethernet-like interfaces and
            sub-interfaces";
       }

       description
         "Augment the generic interface encapsulation with an
          basic 802.1Q VLAN encapsulation for sub-interfaces.";

       /*
        * Matches a single VLAN Id, or pair of VLAN Ids to classify
        * traffic into an L3 service.
        */
       case dot1q-vlan {
         container dot1q-vlan {
           must
             'count(../../if-cmn:forwarding-mode) = 0 or ' +
             'derived-from-or-self(../../if-cmn:forwarding-mode,' +
                                   '"if-cmn:layer-3-forwarding")' {
               error-message
                 "If the interface forwarding-mode leaf is set then it
                  must be set to an identity that derives from
                  layer-3-forwarding";

               description
                 "The forwarding-mode leaf on an interface can
                  optionally be used to enforce consistency of
                  configuration";
             }


           description
             "Match VLAN tagged frames with specific VLAN Ids";
           container outer-tag {
             presence "The outermost VLAN tag exists";
```

```
            description
              "Classifies traffic using the outermost VLAN tag on the
               frame.";

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }

          container second-tag {
            must
              '../outer-tag/tag-type = "dot1q-types:s-vlan" and ' +
              'tag-type = "dot1q-types:c-vlan"' {

              error-message
                "When matching two tags, the outermost tag must be
                 specified and of S-VLAN type and the second outermost
                 tag must be of C-VLAN tag type";

              description
                "For IEEE 802.1Q interoperability, when matching two
                 tags, it is required that the outermost tag exists and
                 is an S-VLAN, and the second outermost tag is a
                 C-VLAN";
            }

            presence "The second outermost VLAN tag exists";

            description
              "Classifies traffic using the second outermost VLAN tag
               on the frame.";

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Flexible Encapsulation YANG Module

This YANG module augments the encapsultion container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

This YANG module also augments the interface container defined in [RFC8343].

```
<CODE BEGINS> file "ietf-flexible-encapsulation@2017-10-30.yang"
module ietf-flexible-encapsulation {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-flexible-encapsulation";

  prefix flex;

  import ietf-interfaces {
    prefix if;
  }

  import iana-if-type {
    prefix ianaift;
  }

  import ietf-interfaces-common {
    prefix if-cmn;
  }

  import ieee802-dot1q-types {
    prefix dot1q-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>

     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 YANG module describes interface configuration for flexible
     VLAN matches and rewrites.";

  revision 2017-10-30 {
    description "Latest draft revision";

    reference
```

```
        "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-03";
    }

    feature flexible-rewrites {
      description
        "This feature indicates whether the network element supports
          specifying flexible rewrite operations";
    }

    feature asymmetric-rewrites {
      description
        "This feature indicates whether the network element supports
         specifying different rewrite operations for the ingress
         rewrite operation and egress rewrite operation.";
    }

    feature dot1q-tag-rewrites {
      description
        "This feature indicates whether the network element supports
         the flexible rewrite functionality specifying flexible 802.1Q
         tag rewrites";
    }

    /*
     * flexible-match grouping.
     *
     * This grouping represents a flexible match.
     *
     * The rules for a flexible match are:
     *     1. default, untagged, priority tag, or a stack of tags.
     *   - Each tag in the stack of tags matches:
     *      1. tag type (802.1Q or 802.1ad) +
     *      2. tag value:
     *        i. single tag
     *        ii. set of tag ranges/values.
     *        iii. "any" keyword
     */
    grouping flexible-match {
      description "Flexible match";
      choice match-type {
        mandatory true;
        description "Provides a choice of how the frames may be
                     matched";

        case default {
          description "Default match";
          leaf default {
            type empty;
```

```
            description
              "Default match.  Matches all traffic not matched to any
               other peer sub-interface by a more specific
               encapsulation.";
          } // leaf default
        } // case default

        case untagged {
          description "Match untagged Ethernet frames only";
          leaf untagged {
            type empty;
            description
              "Untagged match.  Matches all untagged traffic.";
          } // leaf untagged
        } // case untagged

        case dot1q-priority-tagged {
          description
            "Match 802.1Q priority tagged Ethernet frames only";

          container dot1q-priority-tagged {
            description "802.1Q priority tag match";
            leaf tag-type {
              type dot1q-types:dot1q-tag-type;
              description "The 802.1Q tag type of matched priority
                           tagged packets";
            }
          }
        }

        case dot1q-vlan-tagged {
          container dot1q-vlan-tagged {
            description "Matches VLAN tagged frames";

            container outer-tag {
              presence "The outermost VLAN tag exists";

              description
                "Classifies traffic using the outermost VLAN tag on the
                 frame.";

              uses
                'dot1q-types:'+
                'dot1q-tag-ranges-or-any-classifier-grouping';
            }

            container second-tag {
              must
```

```
                '../outer-tag/tag-type = "dot1q-types:s-vlan" and ' +
                'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "When matching two tags, the outermost tag must be
                   specified and of S-VLAN type and the second
                   outermost tag must be of C-VLAN tag type";

                description
                  "For IEEE 802.1Q interoperability, when matching two
                   tags, it is required that the outermost tag exists
                   and is an S-VLAN, and the second outermost tag is a
                   C-VLAN";
              }

              presence "The second outermost VLAN tag exists";

              description
                "Classifies traffic using the second outermost VLAN tag
                 on the frame.";

              uses
                'dot1q-types:'+
                'dot1q-tag-ranges-or-any-classifier-grouping';
            }

            leaf match-exact-tags {
              type empty;
              description
                "If set, indicates that all 802.1Q VLAN tags in the
                 Ethernet frame header must be explicitly matched, i.e.
                 the EtherType following the matched tags must not be a
                 802.1Q tag EtherType.  If unset then extra 802.1Q VLAN
                 tags are allowed.";
            }
          }
        }
      } // encaps-type
    }

    /*
     * Grouping for tag-rewrite that can be expressed either
     * symmetrically, or in the ingress and/or egress directions
     * independently.
     */
    grouping dot1q-tag-rewrite {
      description "Flexible rewrite";
      leaf pop-tags {
```

```
          type uint8 {
            range 1..2;
          }
          description "The number of tags to pop (or translate if used in
                       conjunction with push-tags)";
        }

        container push-tags {
          description "The 802.1Q tags to push (or translate if used in
                       conjunction with pop-tags)";

          container outer-tag {
            presence
              "Indicates existence of the outermost VLAN tag to
               push/rewrite";

            description
              "The outermost VLAN tag to push onto the frame.";

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }

          container second-tag {
            must
              '../outer-tag/tag-type = "dot1q-types:s-vlan" and ' +
              'tag-type = "dot1q-types:c-vlan"' {

              error-message
              "When pushing/rewriting two tags, the outermost tag must be
               specified and of S-VLAN type and the second outermost tag
               must be of C-VLAN tag type";

              description
              "For IEEE 802.1Q interoperability, when pushing two tags,
               it is required that the outermost tag exists and is an
               S-VLAN, and the second outermost tag is a C-VLAN";
            }

            presence
              "Indicates existence of a second outermost VLAN tag to
               push/rewrite.";

            description
              "The second outermost VLAN tag to push onto the frame.";

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }
        }
```

```
    }

    /*
     * Grouping for all flexible rewrites of fields in the L2 header.
     *
     * This currently only includes flexible tag rewrites, but is
     * designed to be extensible to cover rewrites of other fields in
     * the L2 header if required.
     */
    grouping flexible-rewrite {
      description "Flexible rewrite";

      /*
       * Tag rewrite.
       *
       * All tag rewrites are formed using a combination of pop-tags
       * and push-tags operations.
       */
      container dot1q-tag-rewrite {
        if-feature dot1q-tag-rewrites;
        description "Tag rewrite.  Translate operations are expressed
                     as a combination of tag push and pop operations.";
        uses dot1q-tag-rewrite;
      }
    }
    augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
            "if-cmn:encaps-type" {
      when
          "derived-from-or-self(../if:type,
                                'ianaift:ethernetCsmacd') or
           derived-from-or-self(../if:type,
                                'ianaift:ieee8023adLag') or
           derived-from-or-self(../if:type,
                                'if-cmn:ethSubInterface')" {
        description
          "Applies only to Ethernet-like interfaces and
           sub-interfaces";
      }
      description
        "Add flexible match and rewrite for VLAN sub-interfaces";

      /*
       * A flexible encapsulation allows for the matching of ranges and
       * sets of VLAN Ids.  The structure is also designed to be
       * extended to allow for matching/rewriting other fields within
       * the L2 frame header if required.
       */
      case flexible {
```

```
          description "Flexible encapsulation and rewrite";
          container flexible {
              must
                'count(../../if-cmn:forwarding-mode) = 0 or ' +
                'derived-from-or-self(../../if-cmn:forwarding-mode,' +
                                    '"if-cmn:layer-2-forwarding")' {
                error-message
                  "If the interface forwarding-mode leaf is set then it
                   must be set to an identity that derives from
                   layer-2-forwarding";

                description
                  "The forwarding-mode leaf on an interface can
                   optionally be used to enforce consistency of
                   configuration";
              }

            description "Flexible encapsulation and rewrite";

            container match {
              description
                "The match used to classify frames to this interface";
              uses flexible-match;
            }

            container rewrite {
              if-feature flexible-rewrites;
              description "L2 frame rewrite operations";
              choice direction {
                description
                  "Whether the rewrite policy is symmetrical or
                   asymmetrical";
                case symmetrical {
                  container symmetrical {
                    uses flexible-rewrite;
                    description
                      "Symmetrical rewrite.  Expressed in the ingress
                       direction, but the reverse operation is applied to
                       egress traffic";
                  }
                }

                /*
                 * Allow asymmetrical rewrites to be specified.
                 */
                case asymmetrical {
                  if-feature asymmetric-rewrites;
                  description "Asymmetrical rewrite";
```

```
            container ingress {
              uses flexible-rewrite;
              description "Ingress rewrite";
            }
            container egress {
              uses flexible-rewrite;
              description "Egress rewrite";
            }
          }
        }
      }

      /*
       * For encapsulations that match a range of VLANs (or Any),
       * allow configuration to specify the default 802.1Q VLAN tag
       * values to use for any traffic that is locally sourced from
       * an interface on the device.
       */
      container local-traffic-default-encaps {
        presence
          "A local traffic default encapsulation has been
           specified";
        description
          "The 802.1Q VLAN tags to use by default for locally
           sourced traffic";

        container outer-tag {
          presence
            "Indicates existence of the outermost VLAN tag";

          description
            "The outermost VLAN tag for locally sourced traffic";

          uses dot1q-types:dot1q-tag-classifier-grouping;
        }

        container second-tag {
          must
            '../outer-tag/tag-type = "dot1q-types:s-vlan" and ' +
             'tag-type = "dot1q-types:c-vlan"' {

          error-message
            "When specifying two tags, the outermost tag must be
             specified and of S-VLAN type and the second outermost
             tag must be of C-VLAN tag type";

          description
            "For IEEE 802.1Q interoperability, when specifying two
```

```
                   tags, it is required that the outermost tag exists and
                   is an S-VLAN, and the second outermost tag is a
                   C-VLAN";
                }

                presence
                  "Indicates existence of a second outermost VLAN tag.";

                description
                  "The second outermost VLAN tag for locally sourced
                   traffic";

                uses dot1q-types:dot1q-tag-classifier-grouping;
              }
            }
          }
        }
      }
    }
    <CODE ENDS>
```

7. Examples

The following sections give examples of configuring a sub-interface supporting L3 forwarding, and also a sub-interface being used in conjunction with the IETF L2VPN YANG model [I-D.ietf-bess-l2vpn-yang].

7.1. Layer 3 sub-interfaces with IPv6

This example illustrates a layer 3 sub-interface configured to match traffic with a S-VLAN tag of 10, and C-VLAN tag of 20.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
  xmlns:if-cmn="urn:ietf:params:xml:ns:yang:ietf-interfaces-common">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
    </interface>
    <interface>
      <name>eth0.1</name>
```

```
        <type>ianaift:l2vlan</type>
        <if-cmn:parent-interface>eth0</if-cmn:parent-interface>
        <if-cmn:encapsulation>
          <dot1q-vlan
           xmlns="urn:ietf:params:xml:ns:yang:ietf-if-l3-vlan">
            <outer-tag>
              <tag-type>dot1q-types:s-vlan</tag-type>
              <vlan-id>10</vlan-id>
            </outer-tag>
            <second-tag>
              <tag-type>dot1q-types:c-vlan</tag-type>
              <vlan-id>20</vlan-id>
            </second-tag>
          </dot1q-vlan>
        </if-cmn:encapsulation>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
          <forwarding>true</forwarding>
          <address>
            <ip>2001:db8::10</ip>
            <prefix-length>32</prefix-length>
          </address>
        </ipv6>
      </interface>
      <interface>
        <name>eth0.2</name>
        <type>ianaift:l2vlan</type>
        <if-cmn:parent-interface>eth0</if-cmn:parent-interface>
        <if-cmn:encapsulation>
          <dot1q-vlan
           xmlns="urn:ietf:params:xml:ns:yang:ietf-if-l3-vlan">
            <outer-tag>
              <tag-type>dot1q-types:s-vlan</tag-type>
              <vlan-id>11</vlan-id>
            </outer-tag>
          </dot1q-vlan>
        </if-cmn:encapsulation>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
          <forwarding>true</forwarding>
          <address>
            <ip>2001:db8::11</ip>
            <prefix-length>32</prefix-length>
          </address>
        </ipv6>
      </interface>
    </interfaces>
  </config>
```

7.2. Layer 2 sub-interfaces with L2VPN

This example illustrates a layer 2 sub-interface 'eth0.3' configured to match traffic with a S-VLAN tag of 10, and C-VLAN tag of 21; and both tags removed before the traffic is passed off to the L2VPN service.

It also illustrates another sub-interface 'eth1.0' under a separate physical interface configured to match traffic with a C-VLAN of 50, and the tag removed before traffic is given to any service. Sub-interface 'eth1.0' is not currently bound to any service and hence traffic classifed to that sub-interface is dropped.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
  xmlns:if-cmn="urn:ietf:params:xml:ns:yang:ietf-interfaces-common">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
    </interface>
    <interface>
      <name>eth0.3</name>
      <type>ianaift:l2vlan</type>
      <if-cmn:parent-interface>eth0</if-cmn:parent-interface>
      <if-cmn:encapsulation>
        <flexible
    xmlns="urn:ietf:params:xml:ns:yang:ietf-flexible-encapsulation">
          <match>
            <dot1q-vlan-tagged>
              <outer-tag>
                <tag-type>dot1q-types:s-vlan</tag-type>
                <vlan-id>10</vlan-id>
              </outer-tag>
              <second-tag>
                <tag-type>dot1q-types:c-vlan</tag-type>
                <vlan-id>21</vlan-id>
              </second-tag>
            </dot1q-vlan-tagged>
          </match>
          <rewrite>
            <symmetrical>
              <dot1q-tag-rewrite>
                <pop-tags>2</pop-tags>
```

```
                </dot1q-tag-rewrite>
              </symmetrical>
            </rewrite>
          </flexible>
        </if-cmn:encapsulation>
      </interface>
      <interface>
        <name>eth1</name>
        <type>ianaift:ethernetCsmacd</type>
      </interface>
      <interface>
        <name>eth1.0</name>
        <type>ianaift:l2vlan</type>
        <if-cmn:parent-interface>eth0</if-cmn:parent-interface>
        <if-cmn:encapsulation>
          <flexible
      xmlns="urn:ietf:params:xml:ns:yang:ietf-flexible-encapsulation">
            <match>
              <dot1q-vlan-tagged>
                <outer-tag>
                  <tag-type>dot1q-types:c-vlan</tag-type>
                  <vlan-id>50</vlan-id>
                </outer-tag>
              </dot1q-vlan-tagged>
            </match>
            <rewrite>
              <symmetrical>
                <dot1q-tag-rewrite>
                  <pop-tags>1</pop-tags>
                </dot1q-tag-rewrite>
              </symmetrical>
            </rewrite>
          </flexible>
        </if-cmn:encapsulation>
      </interface>
    </interfaces>
    <network-instances
        xmlns="urn:ietf:params:xml:ns:yang:ietf-network-instance">
      <network-instance
       xmlns:l2vpn="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
        <name>p2p-l2-1</name>
        <description>Point to point L2 service</description>
        <l2vpn:type>l2vpn:vpws-instance-type</l2vpn:type>
        <l2vpn:signaling-type>
          l2vpn:ldp-signaling
        </l2vpn:signaling-type>
        <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
          <name>local</name>
```

```
          <ac>
            <name>eth0.3</name>
          </ac>
        </endpoint>
        <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
          <name>remote</name>
          <pw>
            <name>pw1</name>
          </pw>
        </endpoint>
        <vsi-root>
        </vsi-root>
      </network-instance>
    </network-instances>
    <pseudowires
        xmlns="urn:ietf:params:xml:ns:yang:ietf-pseudowires">
      <pseudowire>
        <name>pw1</name>
        <configured-pw>
          <peer-ip>2001:db8::50></peer-ip>
          <pw-id>100</pw-id>
        </configured-pw>
      </pseudowire>
    </pseudowires>
  </config>
```

8. Acknowledgements

The authors would particularly like to thank John Messenger, Glenn Parsons, and Dan Romascanu for their help progressing this draft.

The authors would also like to thank Alex Campbell, Eric Gray, Giles Heron, Marc Holness, Iftekhar Hussain, Neil Ketley, William Lupton, John Messenger, Glenn Parsons, Ludwig Pauwels, Joseph White, Vladimir Vassilev, and members of the IEEE 802.1 WG for their helpful reviews and feedback on this draft.

9. ChangeLog

9.1. WG version -04

- Added examples

9.2. WG version -03

- Fix namespace bug in XPath identity references, removed extraneous 'dot1q-tag' containers.

9.3. WG version -02

- Use explicit containers for outer and inner tags rather than lists.

9.4. WG version -01

- Tweaked the abstract.
- Removed unnecessary feature for the L3 sub-interface module.
- Update the 802.1Qcp type references.
- Remove extra tag container for L3 sub-interfaces YANG.

9.5. Version -04

- IEEE 802.1 specific types have been removed from the draft. These are now referenced from the 802.1Qcp draft YANG modules.
- Fixed errors in the xpath expressions.

9.6. Version -03

- Incorporates feedback received from presenting to the IEEE 802.1 WG.
- Updates the modules for double tag matches/rewrites to restrict the outer tag type to S-VLAN and inner tag type to C-VLAN.
- Updates the introduction to indicate primary use case is for IETF forwarding protocols.
- Updates the objectives to make IEEE 802.1Q bridge interoperability a key objective.

10. 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".

11. 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:

11.1. if-l3-vlan.yang

The nodes in the if-l3-vlan YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/dot1q-vlan, that are sensitive to this are:

- outer-tag/tag-type
- outer-tag/vlan-id
- second-tag/tag-type
- second-tag/vlan-id

11.2. flexible-encapsulation.yang

There are many nodes in the flexible-encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/match, that are sensitive to this are:

- default
- untagged
- dot1q-priority-tagged
- dot1q-priority-tagged/tag-type
- dot1q-vlan-tagged/outer-tag/vlan-type
- dot1q-vlan-tagged/outer-tag/vlan-id
- dot1q-vlan-tagged/second-tag/vlan-type
- dot1q-vlan-tagged/second-tag/vlan-id

There are also many modes in the flexible-encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/rewrite, that are sensitive to this are:

- symmetrical/dot1q-tag-rewrite/pop-tags
- symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- symmetrical/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- symmetrical/dot1q-tag-rewrite/push-tags/second-tag/vlan-id
- asymmetrical/ingress/dot1q-tag-rewrite/pop-tags
- asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

- asymmetrical/egress/dot1q-tag-rewrite/pop-tags
- asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

- outer-tag/vlan-type
- outer-tag/vlan-id
- second-tag/vlan-type
- second-tag/vlan-id

12. References

12.1. Normative References

[I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tsingh@juniper.net, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-05 (work in progress), July 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://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>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

12.2. Informative References

[dot1Qcp] Holness, M., "802.1Qcp Bridges and Bridged Networks - Amendment: YANG Data Model", 2018.

[I-D.ietf-bess-l2vpn-yang] Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-08 (work in progress), February 2018.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <https://www.rfc-editor.org/info/rfc4448>.

[RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <https://www.rfc-editor.org/info/rfc4761>.

[RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <https://www.rfc-editor.org/info/rfc4762>.

[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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group is also developing a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.

- Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.

- Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.

- In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.

- The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single

Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.
- Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.
- Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.
- VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.
- Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2017

C. Wildes, Ed.
Cisco Systems Inc.
K. Koushik, Ed.
Verizon Wireless
March 13, 2017

A YANG Data Model for Syslog Configuration
draft-ietf-netmod-syslog-model-13

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 September 12, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
  1.1. Requirements Language 2
  1.2. Terminology 2
2. Problem Statement 3
3. Design of the Syslog Model 3
  3.1. Syslog Module 5

4. Syslog YANG Module 7
4.1. The ietf-syslog Module 7
5. Usage Examples 21
6. Acknowledgements 22
7. IANA Considerations 23
8. Security Considerations 23
8.1. Resource Constraints 24
8.2. Inappropriate Configuration 24
9. References 24
9.1. Normative References 24
9.2. Informative References 25
Appendix A. Implementor Guidelines 25
Appendix A.1. Extending Facilities 25
Authors' Addresses 26

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, 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 RFC 5424 [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 "originator" is defined in [RFC 5424]: an "originator"

generates syslog content to be carried in a message.

The terms "relay" and "collectors" are as defined in [RFC 5424].

2. Problem Statement

This document defines a YANG [RFC6020] configuration data model that may be used to configure the syslog feature 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 layers [RFC5424] as implemented on the target system.

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. This model is designed to be very simple for maximum flexibility.

Optional features are used to specify functionality that is present in specific vendor configurations.

Syslog consists of originators, and collectors. The following diagram shows syslog messages flowing from an originator, to collectors where 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).

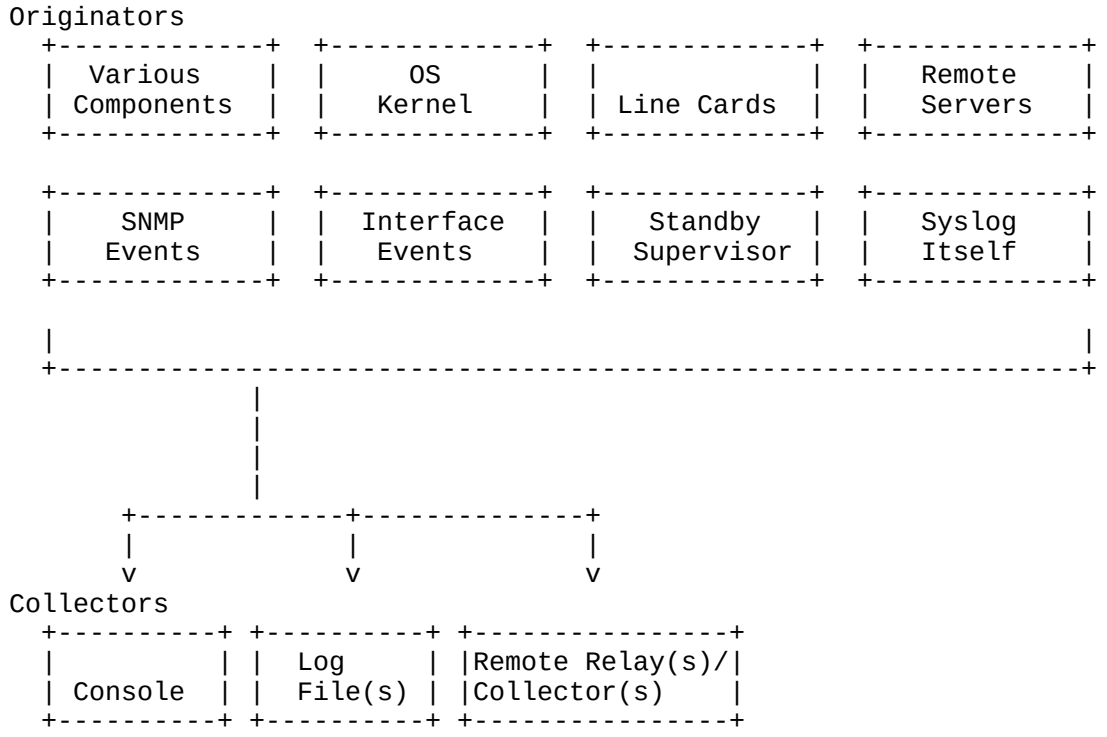


Figure 1. Syslog Processing Flow

The leaves in the syslog model "actions" container correspond to each message collector:

```
console
log file(s)
remote relay(s)/collector(s)
```

Within each action, a selector is used to filter syslog messages. A selector consists of a list of one or more facility-severity matches, and, if supported via the select-match feature, an optional regular expression pattern match that is performed on the SYSLOG-MSG [RFC5424] field.

A syslog message is processed if:

```
There is an element of facility-list (F, S) where
    the message facility matches F (if it is present)
       and the message severity matches S (if it is present)
or the message text matches the regex pattern (if it is present)
```

The facility is one of a specific syslog-facility, or all facilities.

The severity is one of type syslog-severity, all severities, or none. None is a special case that can be used to disable a filter. When filtering severity, the default comparison is that messages of the specified severity and higher are selected to be logged. This is shown in the model as "default equals-or-higher". This behavior can be altered if the select-adv-compare feature is enabled to specify a compare operation and an action. Compare operations are: "equals" to select messages with this single severity, or "equals-or-higher" to select messages of the specified severity and higher. Actions are used to log the message or block the message from being logged.

3.1. Syslog Module

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is defined in [RFC6087].

```
   module: ietf-syslog
      +--rw syslog!
         +--rw actions
            +--rw console! {console-action}?
            |  +--rw selector
            |     +--rw facility-list* [facility severity]
            |     |  +--rw facility            union
            |     |  +--rw severity            union
            |     |  +--rw advanced-compare {select-adv-compare}?
            |     |     +--rw compare?   enumeration
            |     |     +--rw action?    enumeration
            |     +--rw pattern-match?   string {select-match}?
            +--rw file {file-action}?
            |  +--rw log-file* [name]
            |     +--rw name               inet:uri
            |     +--rw selector
            |     |  +--rw facility-list* [facility severity]
            |     |  |  +--rw facility            union
            |     |  |  +--rw severity            union
            |     |  |  +--rw advanced-compare {select-adv-compare}?
            |     |  |     +--rw compare?   enumeration
            |     |  |     +--rw action?    enumeration
            |     |  +--rw pattern-match?   string {select-match}?
            |     +--rw structured-data?   boolean {structured-data}?
            |     +--rw file-rotation
            |        +--rw number-of-files?   uint32 {file-limit-size}?
            |        +--rw max-file-size?     uint32 {file-limit-size}?
            |        +--rw rollover?          uint32 {file-limit-duration}?
            |        +--rw retention?         uint32 {file-limit-duration}?
            +--rw remote {remote-action}?
               +--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
                  +--rw selector
                  |  +--rw facility-list* [facility severity]
                  |  |  +--rw facility            union
                  |  |  +--rw severity            union
                  |  |  +--rw advanced-compare {select-adv-compare}?
                  |  |     +--rw compare?   enumeration
                  |  |     +--rw action?    enumeration
                  |  +--rw pattern-match?   string {select-match}?
                  +--rw structured-data?     boolean {structured-data}?
                  +--rw facility-override?   identityref
                  +--rw source-interface?    if:interface-ref {remote-source-int
erface}?
                  +--rw signing-options! {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
```

Figure 2. ietf-syslog Module Tree

4. Syslog YANG Module

4.1. 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;
  }

  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:   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.

     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.";

  reference
    "RFC 5424: The Syslog Protocol
     RFC 5426: Transmission of Syslog Messages over UDP
     RFC 6587: Transmission of Syslog Messages over TCP
     RFC 5848: Signed Syslog Messages";
```

```
     revision 2017-03-13 {
       description
         "Initial Revision";
       reference
         "RFC XXXX: Syslog YANG Model";
     }

     feature console-action {
       description
         "This feature indicates that the local console action is
          supported.";
     }

     feature file-action {
       description
         "This feature indicates that the local file action is
          supported.";
     }

     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 remote-action {
       description
         "This feature indicates that the remote server action is
          supported.";
     }

     feature remote-source-interface {
       description
         "This feature indicates that source-interface is supported
          supported for the remote-action.";
     }

     feature select-adv-compare {
       description
         "This feature represents the ability to select messages
          using the additional comparison operators 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.";
     }

     typedef syslog-severity {
       type enumeration {
         enum "emergency" {
           value 0;
           description
             "The severity level 'Emergency' indicating that the system
              is unusable.";
         }
         enum "alert" {
           value 1;
           description
             "The severity level 'Alert' indicating that an action must be
              taken immediately.";
         }
         enum "critical" {
           value 2;
           description
             "The severity level 'Critical' indicating a critical condition.";
         }
         enum "error" {
           value 3;
           description
             "The severity level 'Error' indicating an error condition.";
         }
         enum "warning" {
           value 4;
           description
             "The severity level 'Warning' indicating a warning condition.";
         }
         enum "notice" {
           value 5;
           description
             "The severity level 'Notice' indicating a normal but significant
              condition.";
         }
         enum "info" {
           value 6;
           description
             "The severity level 'Info' indicating an informational message.";
         }
```

```
        enum "debug" {
          value 7;
          description
            "The severity level 'Debug' indicating a debug-level message.";
        }
      }
      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.";
    }

    grouping severity-filter {
      description
        "This grouping defines the processing used to select
         log messages by comparing syslog message severity using
         the following processing rules:
          - if 'none', do not match.
          - if 'all', match.
          - else compare message severity with the specified severity
            according to the default compare rule (all messages of the
            specified severity and greater match) or if the
            select-adv-compare feature is present, the advance-compare
            rule.";
      leaf severity {
        type union {
          type syslog-severity;
          type enumeration {
            enum none {
              value -2;
              description
                "This enum describes the case where no severities
                 are selected.";
            }
            enum all {
              value -1;
              description
                "This enum describes the case where all severities
                 are selected.";
            }
          }
        }
        mandatory true;
        description
          "This leaf specifies the syslog message severity.";
      }
      container advanced-compare {
        when '../severity != "all" and
             ../severity != "none"' {
          description
            "The advanced compare container is not applicable for severity
             'all' or severity 'none'";
        }
        if-feature select-adv-compare;
```

```
          leaf compare {
            type enumeration {
              enum equals {
                description
                  "This enum specifies that the severity comparison operation
                   will be equals.";
              }
              enum equals-or-higher {
                description
                  "This enum specifies that the severity comparison operation
                   will be equals or higher.";
              }
            }
            default equals-or-higher;
            description
              "The compare can be used to specify the comparison operator that
               should be used to compare the syslog message severity with the
               specified severity.";
          }
          leaf action {
            type enumeration {
              enum log {
                description
                  "This enum specifies that if the compare operation is true
                   the message will be logged.";
               }
              enum block {
                description
                  "This enum specifies that if the compare operation is true
                   the message will not be logged.";
              }
            }
            default log;
            description
              "The action can be used to spectify if the message should be
               logged or blocked based on the outcome of the compare operation.";
          }
          description
            "This leaf describes additional severity compare operations that can
             be used in place of the default severity comparison. The compare lea
f
             specifies the type of the compare that is done and the action leaf
             specifies the intended result. Example: compare->equals and action->
             no-match means messages that have a severity that is not equal to th
e
             specified severity will be logged.";
        }
      }

      grouping selector {
        description
          "This grouping defines a syslog selector which is used to
           select log messages for the log-action (console, file,
           remote, etc.). Choose one or both of the following:
             facility [<facility> <severity>...]
             pattern-match regular-expression-match-string
```

```
          If both facility and pattern-match are specified, both must
          match in order for a log message to be selected.";
        container selector {
          description
            "This container describes the log selector parameters
             for syslog.";
          list facility-list {
            key "facility severity";
            ordered-by user;
            description
              "This list describes a collection of syslog
               facilities and severities.";
            leaf facility {
              type union {
                type identityref {
                  base 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 severity-filter;
          }
          leaf pattern-match {
            if-feature select-match;
            type string;
            description
              "This leaf describes a Posix 1003.2 regular expression
               string that can be used to select a syslog message for
               logging. 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.
             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 {
        presence "Enables logging.";
        description
          "This container describes the configuration parameters for
           syslog.";
        container actions {
          description
            "This container describes the log-action parameters
             for syslog.";
          container console {
            if-feature console-action;
            presence "Enables logging to the console";
            description
              "This container describes the configuration parameters for
               console logging.";
            uses selector;
          }
          container file {
            if-feature file-action;
            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-rotation {
                description
                  "This container describes the configuration
                   parameters for log file rotation.";
                leaf number-of-files {
                  if-feature file-limit-size;
                  type uint32;
                  default 1;
                  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 uint32;
                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 uint32;
                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 {
          if-feature remote-action;
          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.";
                  }
                }
              }
            }
            uses selector;
            uses structured-data;
            leaf facility-override {
              type identityref {
                base syslog-facility;
              }
```

```
            description
              "If specified, this leaf specifies the facility used
               to override the facility in messages delivered to the
               remote server.";
          }
          leaf source-interface {
            if-feature remote-source-interface;
            type if:interface-ref;
            description
              "This leaf sets the source interface to be used to send
               message to the remote syslog server. 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";
          }
          container signing-options {
            if-feature signed-messages;
            presence
              "If present, syslog-signing options 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;
              units "seconds";
              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;
              units "seconds";
              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;
                  units "seconds";
                  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.";
                }
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

Figure 3. ietf-syslog Module

5. Usage Examples

```
      Requirement:
      Enable console logging of syslogs of severity critical

      Here is the example syslog configuration xml:
      <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>
              <selector>
                <facility-list>
                  <facility>all</facility>
                  <severity>critical</severity>
                </facility-list>
              </selector>
            </console>
          </actions>
        </syslog>
      </config>

      Enable remote logging of syslogs to udp destination 2001:db8:a0b:12f0::1
      for facility auth, severity error

      <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>
                <selector>
                  <facility-list>
                    <facility>auth</facility>
                    <severity>error</severity>
                  </facility-list>
                </selector>
              </destination>
            </remote>
          </actions>
        </syslog>
      </config>
```

Figure 4. ietf-syslog Examples

6. Acknowledgements

The authors wish to thank the following who commented on this proposal:

Andy Bierman

Martin Bjorklund
Alex Campbell
Alex Clemm
Jim Gibson
Jeffrey Haas
John Heasley
Giles Heron
Lisa Huang
Mahesh Jethanandani
Jeffrey K Lange
Jan Lindblad
Chris Lonvick
Tom Petch
Juergen Schoenwaelder
Phil Shafer
Jason Sterne
Peter Van Horne
Kent Watsen
Bert Wijnen
Dale R Worley
Aleksandr Zhdankin

7. IANA Considerations

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

Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-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 administrator to ensure that the configured message flow does not overwhelm system resources.

8.2. Inappropriate Configuration

It is the responsibility of the network administrator 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>.

[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

Appendix 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 {
       prefix syslogtypes;
     }

     organization "Example, Inc.";
     contact
       "Example, Inc.
        Customer Service

        E-mail: syslog-yang@example.com";

     description
       "This module contains a collection of vendor-specific YANG type
        definitions for SYSLOG.";

     revision 2017-03-13 {
       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
Verizon Wireless
500 W Dove Rd.
Southlake, TX 76092
US

Phone: +1 512 650-0210
Email: kirankoushik.agraharasreenivasa@verizonwireless.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2018

C. Wildes, Ed.
Cisco Systems Inc.
K. Koushik, Ed.
Verizon Wireless
March 15, 2018

A YANG Data Model for Syslog Configuration
draft-ietf-netmod-syslog-model-26

Abstract

This document defines a YANG data model for the configuration of a syslog process. It is intended this model be used by vendors who implement syslog in their systems.

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

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
  1.1. Requirements Language 2
  1.2. Terminology 3
  1.3. NDMA Compliance 3

1.4. Editorial Note (To be removed by RFC Editor) 3
2. Design of the Syslog Model 3
2.1. Syslog Module 5
3. Syslog YANG Module 7
3.1. The ietf-syslog Module 8
4. Usage Examples 25
5. Acknowledgements 25
6. IANA Considerations 26
6.1. The IETF XML Registry 26
6.2. The YANG Module Names Registry 26
7. Security Considerations 26
8. References 27
8.1. Normative References 27
8.2. Informative References 29
Appendix A. Implementer Guidelines 29
Appendix A.1. Extending Facilities 29
Appendix A.2. Syslog Terminal Output 30
Appendix A.3. Syslog File Naming Convention 30
Authors' Addresses 31

1. Introduction

This document defines a YANG [RFC7950] configuration data model that may be used to configure the syslog feature 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 layers as implemented on the target system.

Essentially, a syslog process receives messages (from the kernel, processes, applications or other syslog processes) and processes them. The processing may involve logging to a local file, and/or displaying on console, 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.

Such definitions of syslog protocol are defined in [RFC5424], and are used in this RFC.

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

1.1. Requirements Language

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

1.2. Terminology

The term "originator" is defined in [RFC5424]: an "originator" generates syslog content to be carried in a message.

The term "relay" is defined in [RFC5424]: a "relay" forwards messages, accepting messages from originators or other relays and sending them to collectors or other relays

The term "collectors" is defined in [RFC5424]: a "collector" gathers syslog content for further analysis.

The term "action" refers to the processing that takes place for each syslog message received.

1.3. NDMA Compliance

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

1.4. Editorial Note (To be removed by RFC Editor)

This document contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- "I-D.ietf-netconf-keystore" --> the assigned RFC value for draft-ietf-netconf-keystore
- "I-D.ietf-netconf-tls-client-server" --> the assigned RFC value for draft-ietf-netconf-tls-client-server
- "zzzz" --> the assigned RFC value for this draft
- I-D.ietf-netmod-revised-datastores --> the assigned RFC value for draft-ietf-netmod-revised-datastores

2. Design of the Syslog Model

The syslog model was designed by comparing various syslog features

implemented by various vendors' in different implementations.

This document addresses the common leafs between implementations and creates a common model, which can be augmented with proprietary features, if necessary. This model is designed to be very simple for maximum flexibility.

Some optional features are defined in this document to specify functionality that is present in specific vendor configurations.

Syslog consists of originators and collectors. The following diagram shows syslog messages flowing from originators, to collectors where filtering can take place.

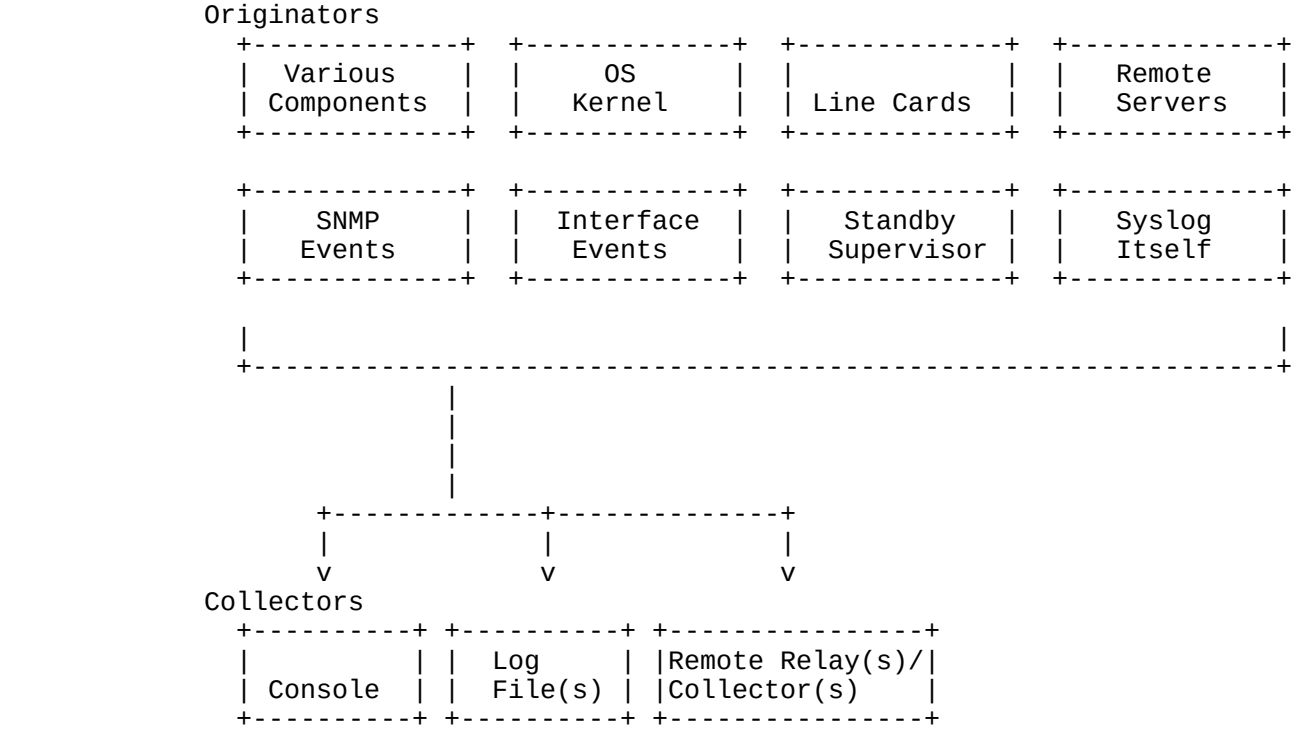


Figure 1. Syslog Processing Flow

Collectors are configured using the leaves in the syslog model "actions" container which correspond to each message collector:

console

log file(s)

remote relay(s)/collector(s)

Within each action, a selector is used to filter syslog messages. A selector consists of a list of one or more filters specified by facility-severity pairs, and, if supported via the select-match feature, an optional regular expression pattern match that is performed on the [RFC5424] field.

A syslog message is processed if:

```
There is an element of facility-list (F, S) where
    the message facility matches F
    and the message severity matches S
and/or the message text matches the regex pattern (if it
    is present)
```

The facility is one of a specific syslog-facility, or all facilities.

The severity is one of type syslog-severity, all severities, or none. None is a special case that can be used to disable a filter. When filtering severity, the default comparison is that messages of the specified severity and higher are selected to be logged. This is shown in the model as "default equals-or-higher". This behavior can be altered if the select-adv-compare feature is enabled to specify a compare operation and an action. Compare operations are: "equals" to select messages with this single severity, or "equals-or-higher" to select messages of the specified severity and higher. Actions are used to log the message or block the message from being logged.

Many vendors extend the list of facilities available for logging in their implementation. An example is included in Extending Facilities (Appendix A.1).

2.1. Syslog Module

A simplified graphical representation of the data model is used in this document. Please see [I-D.ietf-netmod-yang-tree-diagrams] for tree diagram notation.

```
   module: ietf-syslog
      +--rw syslog!
         +--rw actions
            +--rw console! {console-action}?
            |  +--rw facility-filter
            |  |  +--rw facility-list* [facility severity]
            |  |     +--rw facility            union
            |  |     +--rw severity            union
            |  |     +--rw advanced-compare {select-adv-compare}?
            |  |        +--rw compare?   enumeration
            |  |        +--rw action?    enumeration
            |  +--rw pattern-match?     string {select-match}?
            +--rw file {file-action}?
            |  +--rw log-file* [name]
            |     +--rw name                inet:uri
            |     +--rw facility-filter
            |     |  +--rw facility-list* [facility severity]
            |     |     +--rw facility            union
            |     |     +--rw severity            union
            |     |     +--rw advanced-compare {select-adv-compare}?
            |     |        +--rw compare?   enumeration
            |     |        +--rw action?    enumeration
            |     +--rw pattern-match?     string {select-match}?
            |     +--rw structured-data?   boolean {structured-data}?
            |     +--rw file-rotation
            |        +--rw number-of-files?   uint32 {file-limit-size}?
            |        +--rw max-file-size?     uint32 {file-limit-size}?
            |        +--rw rollover?          uint32
            |        |       {file-limit-duration}?
            |        +--rw retention?         uint32
            |                 {file-limit-duration}?
            +--rw remote {remote-action}?
               +--rw destination* [name]
                  +--rw name                 string
                  +--rw (transport)
                  |  +--:(udp)
                  |  |  +--rw udp
                  |  |     +--rw address?   inet:host
                  |  |     +--rw port?      inet:port-number
                  |  +--:(tls)
                  |     +--rw tls
                  |        +--rw address?        inet:host
                  |        +--rw port?           inet:port-number
                  |        +--rw client-auth
                  |        |  +--rw (auth-type)?
                  |        |     +--:(certificate)
                  |        |        +--rw certificate?   leafref
                  |        +--rw server-auth
                  |        |  +--rw pinned-ca-certs?       leafref
                  |        |  +--rw pinned-server-certs?   leafref
                  |        +--rw hello-params
                  |                 {tls-client-hello-params-config}?
                  |           +--rw tls-versions
```

```
|          |  +--rw tls-version*   identityref
|          +--rw cipher-suites
|             +--rw cipher-suite*   identityref
+--rw facility-filter
|  +--rw facility-list* [facility severity]
|     +--rw facility           union
|     +--rw severity           union
|     +--rw advanced-compare {select-adv-compare}?
|        +--rw compare?   enumeration
|        +--rw action?    enumeration
+--rw pattern-match?       string {select-match}?
+--rw structured-data?     boolean {structured-data}?
+--rw facility-override?   identityref
+--rw source-interface?    if:interface-ref
|       {remote-source-interface}?
+--rw signing! {signed-messages}?
   +--rw cert-signers
      +--rw cert-signer* [name]
      |  +--rw name              string
      |  +--rw cert
      |  |  +--rw algorithm?
      |  |  |       identityref
      |  |  +--rw private-key?
      |  |  |       union
      |  |  +--rw public-key?
      |  |  |       binary
      |  |  +---x generate-private-key
      |  |  |  +---w input
      |  |  |     +---w algorithm?
      |  |  |             identityref
      |  |  +--rw certificates
      |  |  |  +--rw certificate* [name]
      |  |  |     +--rw name     string
      |  |  |     +--rw value?   binary
      |  |  +---x generate-certificate-signing-request
      |  |     +---w input
      |  |     |  +---w subject       binary
      |  |     |  +---w attributes?   binary
      |  |     +--ro output
      |  |        +--ro certificate-signing-request
      |  |                binary
      |  +--rw hash-algorithm?   enumeration
      +--rw cert-initial-repeat?   uint32
      +--rw cert-resend-delay?     uint32
      +--rw cert-resend-count?     uint32
      +--rw sig-max-delay?         uint32
      +--rw sig-number-resends?    uint32
      +--rw sig-resend-delay?      uint32
      +--rw sig-resend-count?      uint32
```

Figure 2. ietf-syslog Module Tree

3. Syslog YANG Module

3.1. The ietf-syslog Module

This module imports typedefs from [RFC6991], [I-D.ietf-netmod-rfc7223bis], groupings from [I-D.ietf-netconf-keystore], and [I-D.ietf-netconf-tls-client-server], and it references [RFC5424], [RFC5425], [RFC5426], [RFC5848], [RFC8089], [RFC8174], and [Std-1003.1-2008].

```
   <CODE BEGINS> file "ietf-syslog@2018-03-15.yang"
   module ietf-syslog {
     yang-version 1.1;

     namespace "urn:ietf:params:xml:ns:yang:ietf-syslog";
     prefix syslog;

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

     import ietf-interfaces {
       prefix if;
       reference
         "I-D.ietf-netmod-rfc7223bis: A YANG Data Model
          for Interface Management";
     }

     import ietf-tls-client {
       prefix tlsc;
       reference
         "I-D.ietf-netconf-tls-client-server:
          YANG Groupings for TLS Clients and TLS Servers";
     }

     import ietf-keystore {
       prefix ks;
       reference
         "I-D.ietf-netconf-keystore: YANG Data Model for a
          Keystore Mechanism";
     }

     organization
         "IETF NETMOD (Network Modeling) Working Group";

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

        Editor:   Kiran Agrahara Sreenivasa
                  <mailto:kirankoushik.agraharasreenivasa@
                          verizonwireless.com>

        Editor:   Clyde Wildes
                  <mailto:cwildes@cisco.com>";
     description
       "This module contains a collection of YANG definitions
        for syslog configuration.

        Copyright (c) 2018 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 zzzz
        (http://tools.ietf.org/html/rfczzzz); see the RFC itself for
        full legal notices.";

    revision 2018-03-15 {
      description
        "Initial Revision";
      reference
        "RFC zzzz: Syslog YANG Model";
    }

    feature console-action {
      description
        "This feature indicates that the local console action is
         supported.";
    }

    feature file-action {
      description
        "This feature indicates that the local file action is
         supported.";
    }

    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 remote-action {
      description
        "This feature indicates that the remote server action is
         supported.";
    }
```

```
     feature remote-source-interface {
       description
         "This feature indicates that source-interface is supported
          supported for the remote-action.";
     }

     feature select-adv-compare {
       description
         "This feature represents the ability to select messages
          using the additional comparison operators 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.";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     feature signed-messages {
       description
         "This feature represents the ability to configure signed
          syslog messages.";
       reference
         "RFC 5848: Signed Syslog Messages";
     }

     typedef syslog-severity {
         type enumeration {
         enum "emergency" {
           value 0;
           description
             "The severity level 'Emergency' indicating that the
              system is unusable.";
         }
         enum "alert" {
           value 1;
           description
             "The severity level 'Alert' indicating that an action
              must be taken immediately.";
         }
         enum "critical" {
           value 2;
           description
             "The severity level 'Critical' indicating a critical
              condition.";
```

```
        }
        enum "error" {
          value 3;
          description
            "The severity level 'Error' indicating an error
             condition.";
        }
        enum "warning" {
          value 4;
          description
            "The severity level 'Warning' indicating a warning
             condition.";
        }
        enum "notice" {
          value 5;
          description
            "The severity level 'Notice' indicating a normal but
             significant condition.";
        }
        enum "info" {
          value 6;
          description
            "The severity level 'Info' indicating an informational
             message.";
        }
        enum "debug" {
          value 7;
          description
            "The severity level 'Debug' indicating a debug-level
             message.";
        }
      }
      description
        "The definitions for Syslog message severity.
         Note that a lower value is a higher severity. Comparisons of
         equal-or-higher severity mean equal or lower numeric value";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity syslog-facility {
      description
        "This identity is used as a base for all syslog facilities.";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity kern {
      base syslog-facility;
      description
        "The facility for kernel messages (0).";
      reference
        "RFC 5424: The Syslog Protocol";
    }
```

```
     identity user {
       base syslog-facility;
       description
         "The facility for user-level messages (1).";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     identity mail {
       base syslog-facility;
       description
         "The facility for the mail system (2).";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     identity daemon {
       base syslog-facility;
       description
         "The facility for the system daemons (3).";
       reference
         "RFC 5424: The Syslog Protocol";
      }

     identity auth {
       base syslog-facility;
       description
         "The facility for security/authorization messages (4).";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     identity syslog {
       base syslog-facility;
       description
         "The facility for messages generated internally by syslogd
          facility (5).";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     identity lpr {
       base syslog-facility;
       description
         "The facility for the line printer subsystem (6).";
       reference
         "RFC 5424: The Syslog Protocol";
     }

     identity news {
       base syslog-facility;
       description
         "The facility for the network news subsystem (7).";
```

```
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity uucp {
      base syslog-facility;
      description
        "The facility for the UUCP subsystem (8).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity cron {
      base syslog-facility;
      description
        "The facility for the clock daemon (9).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity authpriv {
      base syslog-facility;
      description
        "The facility for privileged security/authorization messages
         (10).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity ftp {
      base syslog-facility;
      description
        "The facility for the FTP daemon (11).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity ntp {
      base syslog-facility;
      description
        "The facility for the NTP subsystem (12).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity audit {
      base syslog-facility;
      description
        "The facility for log audit messages (13).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity console {
```

```
      base syslog-facility;
      description
        "The facility for log alert messages (14).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity cron2 {
      base syslog-facility;
      description
        "The facility for the second clock daemon (15).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity local0 {
      base syslog-facility;
      description
        "The facility for local use 0 messages (16).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity local1 {
      base syslog-facility;
      description
        "The facility for local use 1 messages (17).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity local2 {
      base syslog-facility;
      description
        "The facility for local use 2 messages (18).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity local3 {
      base syslog-facility;
      description
        "The facility for local use 3 messages (19).";
      reference
        "RFC 5424: The Syslog Protocol";
    }

    identity local4 {
      base syslog-facility;
      description
        "The facility for local use 4 messages (20).";
      reference
        "RFC 5424: The Syslog Protocol";
    }
```

```
identity local5 {
  base syslog-facility;
  description
    "The facility for local use 5 messages (21).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity local6 {
  base syslog-facility;
  description
    "The facility for local use 6 messages (22).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity local7 {
  base syslog-facility;
  description
    "The facility for local use 7 messages (23).";
  reference
    "RFC 5424: The Syslog Protocol";
}

grouping severity-filter {
  description
    "This grouping defines the processing used to select
     log messages by comparing syslog message severity using
     the following processing rules:
      - if 'none', do not match.
      - if 'all', match.
      - else compare message severity with the specified severity
        according to the default compare rule (all messages of the
        specified severity and greater match) or if the
        select-adv-compare feature is present, use the
        advance-compare rule.";
  leaf severity {
    type union {
      type syslog-severity;
      type enumeration {
        enum none {
          value 2147483647;
          description
            "This enum describes the case where no severities
             are selected.";
        }
        enum all {
          value -2147483648;
          description
            "This enum describes the case where all severities
             are selected.";
        }
      }
```

```
          }
          mandatory true;
          description
            "This leaf specifies the syslog message severity.";
        }
        container advanced-compare {
          when '../severity != "all" and
               ../severity != "none"' {
            description
              "The advanced compare container is not applicable for
               severity 'all' or severity 'none'";
          }
          if-feature select-adv-compare;
          leaf compare {
            type enumeration {
              enum equals {
                description
                  "This enum specifies that the severity comparison
                   operation will be equals.";
              }
              enum equals-or-higher {
                description
                  "This enum specifies that the severity comparison
                   operation will be equals or higher.";
              }
            }
            default equals-or-higher;
            description
              "The compare can be used to specify the comparison
               operator that should be used to compare the syslog message
               severity with the specified severity.";
          }
          leaf action {
            type enumeration {
              enum log {
                description
                  "This enum specifies that if the compare operation is
                   true the message will be logged.";
               }
              enum block {
                description
                  "This enum specifies that if the compare operation is
                   true the message will not be logged.";
              }
            }
            default log;
            description
              "The action can be used to specify if the message should
               be logged or blocked based on the outcome of the compare
               operation.";
          }
          description
            "This container describes additional severity compare
             operations that can be used in place of the default
```

```
            severity comparison. The compare leaf specifies the type of
            the compare that is done and the action leaf specifies the
            intended result.
            Example: compare->equals and action->block means
            messages that have a severity that are equal to the
            specified severity will not be logged.";
        }
      }

      grouping selector {
        description
          "This grouping defines a syslog selector which is used to
           select log messages for the log-actions (console, file,
           remote, etc.). Choose one or both of the following:
             facility [<facility> <severity>...]
             pattern-match regular-expression-match-string
           If both facility and pattern-match are specified, both must
           match in order for a log message to be selected.";
        container facility-filter {
          description
            "This container describes the syslog filter parameters.";
          list facility-list {
            key "facility severity";
            ordered-by user;
            description
              "This list describes a collection of syslog
               facilities and severities.";
            leaf facility {
              type union {
                type identityref {
                  base 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 severity-filter;
          }
        }
        leaf pattern-match {
          if-feature select-match;
          type string;
          description
            "This leaf describes a Posix 1003.2 regular expression
             string that can be used to select a syslog message for
             logging. The match is performed on the SYSLOG-MSG field.";
          reference
```

```
            "RFC 5424: The Syslog Protocol
             Std-1003.1-2008 Regular Expressions";
        }
      }

      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.
             If true, messages will be written with one or more
             STRUCTURED-DATA elements; if false, messages will be
             written with STRUCTURED-DATA = NILVALUE.";
          reference
            "RFC 5424: The Syslog Protocol";
        }
      }

      container syslog {
        presence "Enables logging.";
        description
          "This container describes the configuration parameters for
           syslog.";
        container actions {
          description
            "This container describes the log-action parameters
             for syslog.";
          container console {
            if-feature console-action;
            presence "Enables logging to the console";
            description
              "This container describes the configuration parameters
               for console logging.";
            uses selector;
          }
          container file {
            if-feature file-action;
            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:.";
            reference
                "RFC 8089: The file URI Scheme";
          }
          uses selector;
          uses structured-data;
          container file-rotation {
            description
              "This container describes the configuration
               parameters for log file rotation.";
            leaf number-of-files {
              if-feature file-limit-size;
              type uint32;
              default 1;
              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 uint32;
              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 uint32;
              units "minutes";
              description
                "This leaf specifies the length of time that
                 completed/closed log event files should be stored
                 in the file system before they are removed.";
            }
          }
        }
      }
```

```
          container remote {
            if-feature remote-action;
            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 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 ";
                    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 6514;
                    description
                      "TCP port 6514 has been allocated as the default
                       port for syslog over TLS.";
                  }
                  uses tlsc:tls-client-grouping;
                }
              }
            }
            uses selector;
            uses structured-data;
            leaf facility-override {
              type identityref {
                base syslog-facility;
              }
              description
                "If specified, this leaf specifies the facility used
                 to override the facility in messages delivered to
                 the remote server.";
            }
            leaf source-interface {
              if-feature remote-source-interface;
              type if:interface-ref;
              description
                "This leaf sets the source interface to be used to
                 send messages to the remote syslog server. If not
                 set, messages can be sent on any interface.";
            }
            container signing {
              if-feature signed-messages;
              presence
                "If present, syslog-signing options is activated.";
              description
                "This container describes the configuration
                 parameters for signed syslog messages.";
              reference
                "RFC 5848: Signed Syslog Messages";
              container cert-signers {
                description
                  "This container describes the signing certificate
                   configuration for Signature Group 0 which covers
                   the case for administrators who want all Signature
                   Blocks to be sent to a single destination.";
                list cert-signer {
                  key "name";
```

```
                description
                  "This list describes a collection of syslog
                   message signers.";
                leaf name {
                  type string;
                  description
                    "This leaf specifies the name of the syslog
                     message signer.";
                }
                container cert {
                  uses ks:private-key-grouping;
                  uses ks:certificate-grouping;
                  description
                   "This is the certificate that is periodically
                    sent to the remote receiver. Selection of the
                    certificate also implicitly selects the private
                    key used to sign the syslog messages.";
                }
                leaf hash-algorithm {
                  type enumeration {
                    enum SHA1 {
                      value 1;
                      description
                        "This enum describes the SHA1 algorithm.";
                    }
                    enum SHA256 {
                      value 2;
                      description
                        "This enum describes the SHA256 algorithm.";
                    }
                  }
                  description
                    "This leaf describes the syslog signer hash
                     algorithm used.";
                }
              }
              leaf cert-initial-repeat {
                type uint32;
                default 3;
                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 uint32;
                units "seconds";
                default 3600;
                description
                  "This leaf specifies the maximum time delay in
                   seconds until resending the Certificate Block.";
              }
              leaf cert-resend-count {
                type uint32;
```

```
                  default 0;
                  description
                    "This leaf specifies the maximum number of other
                     syslog messages to send until resending the
                     Certificate Block.";
                }
                leaf sig-max-delay {
                  type uint32;
                  units "seconds";
                  default 60;
                  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 uint32;
                  default 0;
                  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 RFC 5426 is used.).";
                }
                leaf sig-resend-delay {
                  type uint32;
                  units "seconds";
                  default 5;
                  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 uint32;
                  default 0;
                  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. A value of 0 means
                     that you don't resend based on the number of
                     messages.";
                }
              }
            }
          }
        }
      }
    }
```

```
   }
   <CODE ENDS>
```

Figure 3. ietf-syslog Module

4. Usage Examples

```
      Requirement:
      Enable console logging of syslogs of severity critical

      <syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
        <actions>
          <console>
            <facility-filter>
              <facility-list>
                <facility>all</facility>
                <severity>critical</severity>
              </facility-list>
            </facility-filter>
          </console>
        </actions>
      </syslog>

      Enable remote logging of syslogs to udp destination
      foo.example.com for facility auth, severity error

      <syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
        <actions>
          <remote>
            <destination>
              <name>remote1</name>
              <udp>
                <address>foo.example.com</address>
              </udp>
              <facility-filter>
                <facility-list>
                  <facility>auth</facility>
                  <severity>error</severity>
                </facility-list>
              </facility-filter>
            </destination>
          </remote>
        </actions>
      </syslog>
```

Figure 4. ietf-syslog Examples

5. Acknowledgements

The authors wish to thank the following who commented on this proposal:

Andy Bierman, Martin Bjorklund, Alex Campbell, Alex Clemm, Francis Dupont, Jim Gibson, Jeffrey Haas, Bob Harold, John Heasley, Giles Heron, Lisa Huang, Mahesh Jethanandani, Warren Kumari, Jeffrey K Lange, Jan Lindblad, Chris Lonvick, Alexey Melnikov, Kathleen Moriarty, Tom Petch, Adam Roach, Juergen Schoenwaelder, Phil Shafer, Yaron Sheffer, Jason Sterne, Peter Van Horne, Kent Watsen, Bert Wijnen, Dale R Worley, and Aleksandr Zhdankin.

6. IANA Considerations

6.1. The IETF XML Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-syslog
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC7895]. Following the format in [RFC7950], the following registration is requested:

```
name:         ietf-syslog
namespace:    urn:ietf:params:xml:ns:yang:ietf-syslog
prefix:       ietf-syslog
reference:    RFC zzzz
```

7. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

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 should be considered sensitive or vulnerable in all network environments. Logging in particular is used to assess the state of systems and can be used to indicate a network compromise. If logging were to be disabled through malicious means, attacks may not be readily detectable. Therefore write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations and on network security.

In addition there are data nodes that require careful analysis and review. These are the subtrees and data nodes and their sensitivity/ vulnerability:

- facility-filter/pattern-match: When writing this node, implementations MUST ensure that the regular expression pattern match is not constructed to cause a regular expression denial of service attack due to a pattern that causes the regular expression implementation to work very slowly (exponentially related to input size).

- remote/destination/signing/cert-signer: When writing this subtree, implementations MUST NOT specify a private key that is used for any other purpose.

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:

- remote/destination/transport: This subtree contains information about other hosts in the network, and the TLS transport certificate properties if TLS is selected as the transport protocol.

- remote/destination/signing: This subtree contains information about the syslog message signing properties including signing certificate information.

There are no RPC operations defined in this YANG module.

8. References

8.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism", Internet-Draft draft-ietf-netconf-keystore-04, October 2017.

[I-D.ietf-netconf-tls-client-server]

Watsen, K. and G. Wu, "YANG Groupings for TLS Clients and TLS Servers", Internet-Draft draft-ietf-netconf-tls-client-server-05, October 2017.

[I-D.ietf-netmod-rfc7223bis]
Bjorklund, M., "A YANG Data Model for Interface Management", Internet-Draft draft-ietf-netmod-rfc7223bis-03, January 2018.

[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, <https://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>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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>.

[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>.

[RFC8089] Kerwin, M., "The "file" URI Scheme", RFC 8089, DOI 10.17487/RFC8089, February 2017, <https://www.rfc-editor.org/info/rfc8089>.

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

[Std-1003.1-2008]

The Open Group, ""Chapter 9: Regular Expressions". The Open Group Base Specifications Issue 6, IEEE Std 1003.1-2008, 2016 Edition.", September 2016, <http://pubs.opengroup.org/onlinepubs/9699919799/>.

8.2. Informative References

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

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", Internet-Draft draft-ietf-netmod-yang-tree-diagrams-06, February 2018.

[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>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

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

Appendix A. Implementer Guidelines

Appendix 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 example-vendor-syslog-types {
     namespace "http://example.com/ns/vendor-syslog-types";
     prefix vendor-syslogtypes;

     import ietf-syslog {
       prefix syslogtypes;
     }

     organization "Example, Inc.";
     contact
       "Example, Inc.
        Customer Service

        E-mail: syslog-yang@example.com";

     description
       "This module contains a collection of vendor-specific YANG type
        definitions for SYSLOG.";

     revision 2017-08-11 {
       description
         "Version 1.0";
       reference
         "Vendor SYSLOG Types: SYSLOG YANG Model";
     }

     identity vendor_specific_type_1 {
       base syslogtypes:syslog-facility;
       description
         "Adding vendor specific type 1 to syslog-facility";
     }

     identity vendor_specific_type_2 {
       base syslogtypes:syslog-facility;
       description
         "Adding vendor specific type 2 to syslog-facility";
     }
   }
```

Appendix A.2. Syslog Terminal Output

Terminal output with requirements more complex than the console subtree currently provides, are expected to be supported via vendor extensions rather than handled via the file subtree.

Appendix A.3. Syslog File Naming Convention

The syslog/file/log-file/file-rotation container contains configuration parameters for syslog file rotation. This section describes how these fields might be used by an implementer to name syslog files in a rotation process. This information is offered as an informative guide only.

When an active syslog file with a name specified by log-file/name, reaches log-file/max-file-size and/or syslog events arrive after the period specified by log-file/rollover, the logging system can close the file, can compress it, and can name the archive file <log-file/name>.0.gz. The logging system can then open a new active syslog file <log-file/name>.

When the new syslog file reaches either of the size limits referenced above, <log-file/name>.0.gz can be renamed <log-file/name>.1.gz and the new syslog file can be closed, compressed and renamed <log-file/name>.0.gz. Each time that a new syslog file is closed, each of the prior syslog archive files named <log-file/name>.<n>.gz can be renamed to <log-file/name>.<n + 1>.gz.

Removal of archive log files could occur when either or both:

- log-file/number-of-files specified - the logging system can create up to log-file/number-of-files syslog archive files after which, the contents of the oldest archived file could be overwritten.

- log-file/retention specified - the logging system can remove those syslog archive files whose file expiration time (file creation time plus the specified log-file/retention time) is prior to the current time.

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
Verizon Wireless
500 W Dove Rd.
Southlake, TX 76092
US

Phone: +1 512 650-0210
Email: kirankoushik.agraharasreenivasa@verizonwireless.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2017

L. Lhotka
CZ.NIC

March 09, 2017

Using Markup in YANG Text Arguments
draft-lhotka-netmod-yang-markup-00

Abstract

This document defines a YANG extension that allows for specifying a media type for text in the arguments of these YANG statements: "contact", "description", "error-message", "organization" and "reference."

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 2
2. Terminology 3
3. Declaring the Media Type of Text Arguments 3
4. YANG Module 4
5. IANA Considerations 6
6. Security Considerations 7
7. References 7
7.1. Normative References 7
7.2. Informative References 7
Author's Address 8

1. Introduction

Descriptions play an important role in YANG data models. They may specify normative semantic constraints that cannot be expressed formally, instructions for implementors of the data model, and other vital information. That is why descriptions often comprise several paragraphs of text and sometimes also other structures such all numbered or bulleted lists.

So far, YANG modules have mostly used only plain text in the argument of the "description" statement, i.e., formatted text structures using just whitespace. This is generally sufficient for human readers and also for including YANG modules in RFC documents, but less so for tools that are designed to process or use YANG modules in other ways, for example:

- converting YANG compact syntax to other formats (YIN, HTML)
- including YANG modules in standards and other documents whose source form in not plain text
- including description text in user interface elements.

The above considerations also apply, albeit to a lesser extent, to other YANG statements with more structured arguments: "contact", "error-message", "organization" and "reference".

In principle, it is possible to use any kind of markup such as markdown [RFC7764] but doing so would not be not very effective and tool-friendly unless the markup format is either standardized or declared in the module source.

This document defines a YANG extension statement, "text-media-type", that can be placed as a substatement of the "module" or "submodule" statement to indicate the media type that is used throughout the

(sub)module for markup in the arguments of the five YANG statements mentioned above.

2. Terminology

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

3. Declaring the Media Type of Text Arguments

Section 4 below contains the "ietf-yang-text-media-type" module that defines a YANG language extension: "text-media-type". Its purpose is to declare media type that is used for markup in the arguments of the following YANG statements:

- contact
- description
- error-message
- organization
- reference

The common characteristic of these five statements is that their argument is represented as the <text> element in YIN format (Section 13 of [RFC7950]).

The "text-media-type" extension is intended to be used at the top level of a YANG module or submodule, i.e., as a substatement of either "module" or "submodule" statement. The declared media type applies to arguments of the above YANG statements throughout the module or submodule.

The "text-media-type" extension MAY be ignored, and a module in which it appears MUST be valid YANG.

The argument of the "text-media-type" extension is a string specifying the media type in the format defined in [RFC6838]. The media type SHOULD be registered by IANA in the "text" registry [IANA-MEDIA-TYPES]. Media type parameters MAY be used.

It is RECOMMENDED to use only media types representing "lightweight" markup that is easy to read even in the unprocessed source form, such as "text/markdown".

YANG modules have to be use the UTF-8 character encoding, see Section 6 of [RFC7950]. This implies the following rules:

- Whenever the "charset" parameter is present, its value MUST be "UTF-8".
- If no default value is defined for the "charset" parameter of a given media type, or if a default value is defined but is different from "UTF-8", then the "charset" parameter MUST be present with the value of "UTF-8".

Even if the "text-media-type" extension statement is present and markup used in text arguments, all YANG and YIN syntax rules still apply.

The "ietf-yang-text-media-type" module not only defines but also uses the "text-media-type" extension statement, which illustrates its typical use.

4. YANG Module

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

<CODE BEGINS> file "ietf-yang-text-media-type@2017-03-09.yang"

```
module ietf-yang-text-media-type {

  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-text-media-type";

  prefix ymt;

  ymt:text-media-type "text/markdown; charset=UTF-8";

  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>";

  description
    "This module defines the *text-media-type* extension that allows
     for specifying media-type for markup that is used in arguments
     of these YANG statements: contact, description, error-message,
     organization, and reference.

     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 2017-03-09 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: Using Markup in YANG Text Arguments";
  }

  extension text-media-type {
    argument type;
    description
      "This extension allows for specifying a media type that is used
       for markup in the arguments of the following YANG statements:

       - contact

       - description

       - error-message

       - organization
```

```
       - reference

       The *text-media-type* extension statement MAY be used at the
       top level of a module or submodule, i.e., as a substatement of
       *module* or *submodule*, and no more than once. The declared
       media type applies throughout the module or submodule.

       The argument SHOULD be a media type registered by IANA in the
       *text* registry. Media type parameters MAY be present.

       This YANG extension is only indicative and optional to
       implement. Tools MAY ignore it completely or support just a
       subset of markup directives that are available for a given
       media type.";
    reference
      "- IANA: Media Types, 2016-12-21. Available [online]
         (http://www.iana.org/assignments/media-types/media-types.xhtml)

       - [RFC 6838](https://tools.ietf.org/html/rfc6838): Media Type
         Specifications and Registration Procedures";
  }
}

<CODE ENDS>
```

5. 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 the following namespace URI in the "IETF XML registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-text-media-type
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name:	ietf-yang-text-media-type
Namespace:	urn:ietf:params:xml:ns:yang:ietf-yang-text-media-type
Prefix:	ymt
Reference:	RFC XXXX

6. Security Considerations

The "text-media-type" extension defined in this document provides information that is completely optional. YANG modules and submodules in which it is present have to satisfy all rules of the YANG language. The extension therefore doen’t introduce any new threats.

7. References

7.1. Normative References

[IANA-MEDIA-TYPES]
Internet Assigned Numbers Authority, "Media Types", 12 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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[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>.

7.2. Informative References

[RFC7764] Leonard, S., "Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registrations", RFC 7764, DOI 10.17487/RFC7764, March 2016, <http://www.rfc-editor.org/info/rfc7764>.

Author’s Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2017

X. Liu
Jabil
I. Bryskin
Huawei Technologies
V. Beeram
Juniper Networks
T. Saad
Cisco Systems Inc
H. Shah
Ciena
O. Gonzalez de Dios
Telefonica

March 9, 2017

A YANG Data Model for Configuration Scheduling
draft-liu-netmod-yang-schedule-03

Abstract

This document describes a data model for configuration scheduling.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

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

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction.....2
   1.1. Terminology.....3
2. Motivation.....3
3. Configuration Scheduling YANG Data Model Overview.....3
4. Usage Example.....4
5. Relations to Datastores.....7
   5.1. Validation.....7
   5.2. Schedules Expansion and Operational States.....7
   5.3. Server Executions at Scheduled Moments.....7
   5.4. Interactions with Locks.....8
   5.5. Interactions with Authorization Mechanism.....8
6. Synchronization Aspects.....8
7. Configuration Scheduling YANG Module.....8
8. Security Considerations.....14
9. Contributors.....15
10. References.....15
   10.1. Normative References.....15
   10.2. Informative References.....16

1. Introduction

This document introduces a YANG [RFC6020] data model for configuration scheduling. This model can be used together with other YANG data models to specify a schedule applied on a configuration data node, so that the configuration data can take effect according to the schedule. Such a configuration schedule can be one-time or recurring, with its properties persistently saved in the datastores of the management system server.

The mechanism described in this document is designed to complement the one described in [RFC7758], which defines a capability extension to NETCONF to allow time-triggered RPCs. Such RPCs can be executed at a future time moment, but cannot be repeated and is not saved in the persistent datastores.

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].

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

- augment
- data model
- data node

2. Motivation

Some applications benefit from resource scheduling to allow operators to plan ahead of time. Traffic engineering is one of such examples [RFC7399]. When configuration and state models are designed for such applications, it has been considered that certain data objects need to be configured according to predefined schedules. In other situations, operators need to de-configure certain data objects at predefined schedules for the purposes such as maintenance. These data objects are interpreted and implemented by the applicable applications.

Delay/Disruption Tolerant Networking (DTN) is another example for which the scheduled configuration can be used, where a long-lived, reliable, low-latency sequenced data delivery session is unsustainable. Section 4.3 of [I-D.birrane-dtn-ama] describes the Autonomous Parameterized Control. Time-based event is one of the two types of triggers in such a system.

3. Configuration Scheduling YANG Data Model Overview

This document defines a YANG data model that specifies configuration schedules for other YANG data models. For each targeted configuration data object or a group of configuration data objects, an entry is

specified along with requested schedules using this configuration schedule model. The application implementing the targeted schema nodes implements the configuration schedules, configuring or de-configuring the specified objects according to the specified schedules. The model schema of the targeted application does not need changes, so the data model described in this document can be used for any data model. The configuration scheduling YANG data model has the following structure:

```
module: ietf-schedule
   +--rw configuration-schedules
      +--rw target* [object]
         +--rw object        yang:xpath1.0
         +--rw operation?    operation
         +--rw data-value?   anydata
         +--rw schedules
         |  +--rw schedule* [schedule-id]
         |     +--rw schedule-id            uint32
         |     +--rw inclusive-exclusive?   enumeration
         |     +--rw start?                 yang:date-and-time
         |     +--rw schedule-duration?     string
         |     +--rw repeat-interval?       string
         +--ro state
         |  +--ro future-executions
         |     +--ro execution* [start]
         |        +--ro start        yang:date-and-time
         |        +--ro duration?    string
         |        +--ro operation?   operation
         +---n execution
            +---- operation    operation
            +---- datetime?    yang:date-and-time
            +---- results?     anydata
```

4. Usage Example

The following model defines a list of TE (Traffic Engineering) links which can be configured with specified schedules:

```
module: example
   +--rw te-links
      +--rw te-link* [id]
         +--rw id         string
```

```
           +--rw enabled?   boolean
```

The following configuration requests that

- link-1 is configured weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.
- link-2 is de-configured for two hours, starting from 2016-09-15T01:00:00.00Z.

```
<configuration-schedules>
  <target xmlns:ex=""urn:example">
    <object>/ex:te-links</object>
    <operation>configure</operation>
    <data-value>
      <te-link>
   <id>link-1</id>
   <enabled>true</enabled>
      </te-link>
    </data-value>
    <schedules>
      <schedule>01
   <schedule-id>11<schedule-id>
   <start>2016-09-12T23:20:50.52Z</start>
   <schedule-duration>P1D</schedule-duration>
   <repeat-interval>R5/P1W</repeat-interval>
      </schedule>
    </schedules>
  </target>
  <target xmlns:ex=""urn:example">
    <object>/ex:te-links</object>
    <operation>configure</operation>
    <data-value>
      <te-link>
   <id>link-2</id>
   <enabled>true</enabled>
      </te-link>
    </data-value>
    <schedules>
      <schedule>
```

```
      <schedule-id>12<schedule-id>
      <inclusive-exclusive>exclusive</inclusive-exclusive>
      <start>2016-09-15T01:00:00.00Z</start>
      <schedule-duration>P2H</schedule-duration>
         </schedule>
       </schedules>
     </target>
   </configuration-schedules>
```

The following configuration requests that

- link-1 is enabled weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.
- link-2 is not enabled for two hours, starting from 2016-09-15T01:00:00.00Z.

```
   <configuration-schedules>
     <target xmlns:ex=""urn:example">
       <object>/ex:te-links/ex:te-link[ex:link-id='link-
   1']/ex:enabled</object>
       <operation>set</operation>
       <data-value>true</data-value>
       <schedules>
         <schedule>
      <schedule-id>11<schedule-id>
      <start>2016-09-12T23:20:50.52Z</start>
      <schedule-duration>P1D</schedule-duration>
      <repeat-interval>R5/P1W</repeat-interval>
         </schedule>
       </schedules>
     </target>
     <target xmlns:ex=""urn:example">
       <object>/ex:te-links/ex:te-link[ex:link-id='link-
   2']/ex:enabled</object>
       <operation>set</operation>
       <data-value>true</data-value>
       <schedules>
         <schedule>
      <schedule-id>12<schedule-id>
      <inclusive-exclusive>exclusive</inclusive-exclusive>
```

```
      <start>2016-09-15T01:00:00.00Z</start>
      <schedule-duration>P2H</schedule-duration>
         </schedule>
       </schedules>
     </target>
   </configuration-schedules>
```

5. Relations to Datastores

NETCONF defines configuration datastores and operations that can be used to access these datastores. The configuration data encoded according to this data model is persistently saved in the proper datastores in the same way as other data model, such as ietf-interfaces.

5.1. Validation

When configuration data based on this model is received, the server MUST perform syntax validations on the received data nodes, and examine the requested schedules. The server does not validate whether requested target configuration data can be applied to the target configuration objects, until the actual scheduled time arrives.

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects. The server MUST perform the validations on the target configuration data along with the current target configuration objects in the proper datastore.

5.2. Schedules Expansion and Operational States

The server SHOULD expand these schedules and expose them to the client as operational states.

5.3. Server Executions at Scheduled Moments

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects, as if an RPC request is newly received. Whether such a time-triggered configuration is successfully applied depends on the configuration data of the target object and requested configuration data. The results of such executions are sent to the client through notifications. The notification management mechanism described in [I-D.ietf-netconf-yang-push] and [I-D.ietf-netconf-rfc5277bis] can be

used to enable, disable, subscribe, filter, and replay the notifications.

5.4. Interactions with Locks

The rules of datastore lock specified by NETCONF [RFC6241] are checked when the schedule configuration data is received and when the target configuration data is applied.

5.5. Interactions with Authorization Mechanism

If the server implements any authorization mechanism, the authorization rules MUST be checked against this data model schema when the schedule configuration data is received. At each scheduled time moment, the authorization rules MUST be checked against the target objects by using the target configuration data. To check the authorization rules, the server uses the same client credential learned when the initial configuration data was received.

6. Synchronization Aspects

The scheduling mechanisms described in this document assume that servers have access to the wall-clock time. Thus, servers are required to acquire the time-of-day from an external time source, for example using the Network Time Protocol [RFC5905], or the Precision Time Protocol [IEEE1588].

It is assumed that the client and servers rely on a common time source, so as to guarantee that schedules are defined with respect to a common reference. In order to avoid the potential ambiguity of different time zones and daylight saving time, it is recommended to define all schedules in the UTC time zone, using the suffix 'Z'. For example, the time 2016-09-12T23:20:50.52Z, is specified with respect to the UTC time zone.

7. Configuration Scheduling YANG Module

```
<CODE BEGINS> file "ietf-schedule@2017-03-06.yang"
module ietf-schedule {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schedule";

  prefix "sch";
```

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

organization "TBD";
contact "TBD";
description
  "The model allows time scheduling parameters to be specified.";

revision "2017-03-06" {
  description "Initial revision";
  reference "TBD";
}

/*
 * Typedefs
 */
typedef operation {
  type enumeration {
    enum configure {
      description
      "Create the configuration data.";
    }
    enum deconfigure {
      description
      "Remove the configuration data.";
    }
    enum set {
      description
      "Set the specified configuration data.";
    }
    enum reset {
      description
      "Revert the specified configuration data back to the
       original value.";
    }
  }
  description "Operation type.";
}
```

```
     /*
      * Groupings
      */

     grouping schedule-config-attributes {
       description
         "A group of attributes for a schedule.";

       leaf inclusive-exclusive {
         type enumeration {
           enum inclusive {
             description
               "The schedule element is inclusive, i.e., the schedule
                specifies the time at which the element is enabled.";
           }
           enum exclusive {
             description
               "The schedule element is exclusive. i.e., the schedule
                specifies the time at which the element is disabled.";
           }
         }
         default "inclusive";
         description
           "Whether the list item is inclusive or exclusive.";
       }
       leaf start {
         type yang:date-and-time;
         description "Start time.";
       }
       leaf schedule-duration {
         type string {
           pattern
             'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
         }
         description "Schedule duration in ISO 8601 format.";
       }
       leaf repeat-interval {
         type string {
           pattern
             'R\d*/P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?'
```

```
            + '(\d+S)?';
        }
        description "Repeat interval in ISO 8601 format.";
      }
    } // schedule-config-attributes

    grouping schedule-config-notification {
      description
        "A group of attributes for a schedule notification.";

      notification execution {
        description
          "Notification event for an execution performed on a target
           object.";
        leaf operation {
          type operation;
          mandatory true;
          description "Operation type.";
        }
        leaf datetime {
          type yang:date-and-time;
          description
            "The date and time when the execution was performed.";
        }
        anydata results {
          description
            "This chunk of data contains the results of the execution
             performed on the target object. The results are the same
             or equivalent to the contents of a <rpc-reply> message,
             Because of the nature of such a target execution, a
             <rpc-reply> message is not used to return the execution
             results. Instead, this notification is used to serve
             the same purpose.";
        }
      }
    } // schedule-config-notification

    grouping schedule-state-attributes {
      description
        "State attributes for a schedule.";
```

```
       container future-executions {
         description
           "The state information of the nexte scheduled event.";
         list execution {
           key "start";
           description
             "List of scheduled future executions.";
           leaf start {
             type yang:date-and-time;
             description "Start time.";
           }
           leaf duration {
             type string {
               pattern
                 'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
             }
             description "Schedule duration in ISO 8601 format.";
           }
           leaf operation {
             type operation;
             description "Operation type.";
           }
         } // event
       } // future-events
     } // schedule-state-attributes

     grouping schedules {
       description
         "A list of schedules defining when a particular
          configuration takes effect.";
       container schedules {
         description
           "Container of a schedule list defining when a particular
            configuration takes effect.";
         list schedule {
           key "schedule-id";
           description "A list of schedule elements.";
           leaf schedule-id {
             type uint32;
             description "Identifies the schedule element.";
```

```
          }
          uses schedule-config-attributes;
        }
      }
    } // schedules

    /*
     * Configuration data and operational state nodes
     */
    container configuration-schedules {
      description
        "Serves as top-level container for a list of configuration
         schedules.";
      list target {
        key "object";
        description
          "A list of targets that configuration schedules are
           applied.";
        leaf object {
          type yang:xpath1.0;
          description
            "Xpath defining the data items of interest.";
        }
        leaf operation {
          type operation;
          default "configure";
          description
            "Operation type.";
        }
        anydata data-value {
          description
            "The data value applied to the leaf data node
             specified by data-objects.
             The format of the data value depends on the value of the
             leaf operation defined above:
             configure:   data-value is the sub-tree added to the
                          target object;
             deconfigure: data-value is the child to be deleted from
                          the target object;
             set:         the target object MULST be a leaf, and
```

```
                          data-value is the new value to be set to
                          the target object;
               reset:     data-value is ignored.";
          }
          uses schedules;
          container state {
            config false;
            description
              "Operational state data.";
            uses schedule-state-attributes;
          } // state

          uses schedule-config-notification;
        } // target
      } // configuration-schedules
    }
    <CODE ENDS>
```

8. Security Considerations

The configuration, state, action and notification data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (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 contents.

The functionality defined in this memo can potentially allow network reconnaissance; by gathering information about schedules an attacker can learn about the network policy, its temporal behavior, and future events.

The schedule YANG model defines schedules that are writable, creatable, and deletable. Therefore, this model may be considered sensitive or vulnerable in some network environments. An attacker may maliciously configure a schedule in a way that disrupts the normal behavior of the network. Furthermore, an attacker may attempt to maliciously set a schedule or a set of schedules in a way that amplifies an attack, or schedules an attack to a particularly sensitive time instant.

The use of configuration scheduling implicitly assumes that there is an underlying synchronization or time distribution mechanism. Therefore, an attack on the synchronization mechanism may compromise the configuration scheduling. The security considerations of time protocols are discussed further in [RFC 7384].

9. Contributors

Tal Mizrahi

Email: talmi@marvell.com

10. References

10.1. Normative References

[IEEE1588] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Version 2", IEEE Standard 1588.

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

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

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

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

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, October 2014.

[RFC7399] Farrel, A. and King, D., "Unanswered Questions in the Path Computation Element Architecture", RFC 7399, October 2014.

[RFC7758] Mizrahi, T. and Moses, Y., "Time Capability in NETCONF", RFC7758, February 2016.

[I-D.birrane-dtn-ama] Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-04 (work in progress), October 2016. [I-D.ietf-netconf-yang-push] Clemm, A., "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push (Work in Progress).

[I-D.ietf-netconf-rfc5277bis] Clemm, A., "Subscribing to Event Notifications", draft-ietf-netconf-rfc5277bis (Work in Progress).

10.2. Informative References

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean, VA 22102
USA

Email: Xufeng_Liu@jabil.com

Igor Bryskin
Huawei Technologies
Email: Igor.Bryskin@huawei.com

Vishnu Pavan Beeram
Juniper Networks
Email: vbeeram@juniper.net

Tarek Saad
Cisco Systems Inc
Email: tsaad@cisco.com

Himanshu Shah
Ciena
Email: hshah@ciena.com

Oscar Gonzalez de Dios
Telefonica
Email: oscar.gonzalezdedios@telefonica.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2018

X. Liu
Jabil
I. Bryskin
Huawei Technologies
V. Beeram
Juniper Networks
T. Saad
Cisco Systems Inc
H. Shah
Ciena
O. Gonzalez de Dios
Telefonica
March 1, 2018

A YANG Data Model for Configuration Scheduling
draft-liu-netmod-yang-schedule-05

Abstract

This document describes a data model for configuration scheduling.

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 2, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Motivation 3
3. Configuration Scheduling YANG Data Model Overview 3
4. Usage Example 4
5. Relations to Datastores 6
5.1. Validation 6
5.2. Schedules Expansion and Operational States 7
5.3. Server Executions at Scheduled Moments 7
5.4. Interactions with Locks 7
5.5. Interactions with Authorization Mechanism 7
6. Synchronization Aspects 7
7. Configuration Scheduling YANG Module 8
8. IANA Considerations 13
9. Security Considerations 14
10. Contributors 14
11. References 14
11.1. Normative References 14
11.2. Informative References 16
Authors' Addresses 16

1. Introduction

This document introduces a YANG [RFC6020] [RFC7950] data model for configuration scheduling. This model can be used together with other YANG data models to specify a schedule applied on a configuration data node, so that the configuration data can take effect according to the schedule. Such a configuration schedule can be one-time or recurring, with its properties persistently saved in the datastores of the management system server.

The mechanism described in this document is designed to complement the one described in [RFC7758], which defines a capability extension to NETCONF to allow time-triggered RPCs. Such RPCs can be executed at a future time moment, but cannot be repeated and is not saved in the persistent datastores.

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].

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

- augment
- data model
- data node

2. Motivation

Some applications benefit from resource scheduling to allow operators to plan ahead of time. Traffic engineering is one of such examples [RFC7399]. When configuration and state models are designed for such applications, it has been considered that certain data objects need to be configured according to predefined schedules. In other situations, operators need to deconfigure certain data objects at predefined schedules for the purposes such as maintenance. These data objects are interpreted and implemented by the applicable applications.

Delay/Disruption Tolerant Networking (DTN) is another example for which the scheduled configuration can be used, where a long-lived, reliable, low-latency sequenced data delivery session is unsustainable. Section 4.3 of [I-D.birrane-dtn-ama] describes the Autonomous Parameterized Control. Time-based event is one of the two types of triggers in such a system.

3. Configuration Scheduling YANG Data Model Overview

This document defines a YANG data model that specifies configuration schedules for other YANG data models. For each targeted configuration data object or a group of configuration data objects, an entry is specified along with requested schedules using this configuration schedule model. The application implementing the targeted schema nodes implements the configuration schedules, configuring or deconfiguring the specified objects according to the specified schedules. The model schema of the targeted application does not need changes, so the data model described in this document can be used for any data model. The configuration scheduling YANG data model has the following structure:

```
module: ietf-schedule
    +--rw configuration-schedules
       +--rw target* [object]
          +--rw object       yang:xpath1.0
          +--rw schedules
          |  +--rw schedule* [schedule-id]
          |     +--rw schedule-id            uint32
          |     +--rw inclusive-exclusive?   enumeration
          |     +--rw start?                 yang:date-and-time
          |     +--rw schedule-duration?     string
          |     +--rw repeat-interval?       string
          |     +--rw operation?             operation
          |     +--rw data-value?            <anydata>
          +--ro state
          |  +--ro future-executions
          |     +--ro execution* [start]
          |        +--ro start        yang:date-and-time
          |        +--ro duration?    string
          |        +--ro operation?   operation
          +---n execution
             +---- operation    operation
             +---- datetime?    yang:date-and-time
             +---- results?     <anydata>
```

4. Usage Example

The following model defines a list of TE (Traffic Engineering) links which can be configured with specified schedules:

```
module: example
   +--rw te-links
      +--rw te-link* [id]
         +--rw id         string
         +--rw enabled?   boolean
```

The following configuration requests that

- link-1 is configured weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.
- link-2 is deconfigured for two hours, starting from 2016-09-15T01:00:00.00Z.

```
<configuration-schedules>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links</object>
    <schedules>
      <schedule>
        <schedule-id>11<schedule-id>
        <start>2016-09-12T23:20:50.52Z</start>
        <schedule-duration>P1D</schedule-duration>
        <repeat-interval>R5/P1W</repeat-interval>
        <operation>configure</operation>
        <data-value>
          <te-link>
            <id>link-1</id>
            <enabled>true</enabled>
          </te-link>
        </data-value>
      </schedule>
    </schedules>
  </target>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links</object>
    <schedules>
      <schedule>
        <schedule-id>12<schedule-id>
        <inclusive-exclusive>exclusive</inclusive-exclusive>
        <start>2016-09-15T01:00:00.00Z</start>
        <schedule-duration>P2H</schedule-duration>
        <operation>configure</operation>
        <data-value>
          <te-link>
            <id>link-2</id>
            <enabled>true</enabled>
          </te-link>
        </data-value>
      </schedule>
    </schedules>
  </target>
</configuration-schedules>
```

The following configuration requests that

- link-1 is enabled weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.

- link-2 is not enabled for two hours, starting from 2016-09-15T01:00:00.00Z.

```
<configuration-schedules>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links/ex:te-link[ex:link-id='link-1']/ex:enabled
    </object>
    <schedules>
      <schedule>
        <schedule-id>11<schedule-id>
        <start>2016-09-12T23:20:50.52Z</start>
        <schedule-duration>P1D</schedule-duration>
        <repeat-interval>R5/P1W</repeat-interval>
        <operation>set</operation>
        <data-value>true</data-value>
      </schedule>
    </schedules>
  </target>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links/ex:te-link[ex:link-id='link-2']/ex:enabled
    </object>
    <schedules>
      <schedule>
        <schedule-id>12<schedule-id>
        <inclusive-exclusive>exclusive</inclusive-exclusive>
        <start>2016-09-15T01:00:00.00Z</start>
        <schedule-duration>P2H</schedule-duration>
        <operation>set</operation>
        <data-value>true</data-value>
      </schedule>
    </schedules>
  </target>
</configuration-schedules>
```

5. Relations to Datastores

NETCONF defines configuration datastores and operations that can be used to access these datastores. The configuration data encoded according to this data model is persistently saved in the proper datastores in the same way as other data model, such as ietf-interfaces.

5.1. Validation

When configuration data based on this model is received, the server MUST perform syntax validations on the received data nodes, and examine the requested schedules. The server does not validate whether requested target configuration data can be applied to the

target configuration objects, until the actual scheduled time arrives.

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects. The server MUST perform the validations on the target configuration data along with the current target configuration objects in the proper datastore.

5.2. Schedules Expansion and Operational States

The server SHOULD expand these schedules and expose them to the client as operational states.

5.3. Server Executions at Scheduled Moments

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects, as if an RPC request is newly received. Whether such a time-triggered configuration is successfully applied depends on the configuration data of the target object and requested configuration data. The results of such executions are sent to the client through notifications. The notification management mechanism described in [I-D.ietf-netconf-yang-push] and [I-D.ietf-netconf-subscribed-notifications] can be used to enable, disable, subscribe, filter, and replay the notifications.

5.4. Interactions with Locks

The rules of datastore lock specified by NETCONF [RFC6241] are checked when the schedule configuration data is received and when the target configuration data is applied.

5.5. Interactions with Authorization Mechanism

If the server implements any authorization mechanism, the authorization rules MUST be checked against this data model schema when the schedule configuration data is received. At each scheduled time moment, the authorization rules MUST be checked against the target objects by using the target configuration data. To check the authorization rules, the server uses the same client credential learned when the initial configuration data was received.

6. Synchronization Aspects

The scheduling mechanisms described in this document assume that servers have access to the wall-clock time. Thus, servers are required to acquire the time-of-day from an external time source, for

example using the Network Time Protocol [RFC5905], or the Precision Time Protocol [IEEE1588].

It is assumed that the client and servers rely on a common time source, so as to guarantee that schedules are defined with respect to a common reference. In order to avoid the potential ambiguity of different time zones and daylight saving time, it is recommended to define all schedules in the UTC time zone, using the suffix 'Z'. For example, the time 2016-09-12T23:20:50.52Z, is specified with respect to the UTC time zone.

7. Configuration Scheduling YANG Module

```
<CODE BEGINS> file "ietf-schedule@2018-02-26.yang"
module ietf-schedule {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schedule";

  prefix "sch";

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

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     Editor:   Xufeng Liu
               <mailto:Xufeng_Liu@jabil.com>

     Editor:   Igor Bryskin
               <mailto:Igor.Bryskin@huawei.com>

     Editor:   Vishnu Pavan Beeram
               <mailto:vbeeram@juniper.net>

     Editor:   Tarek Saad
               <mailto:tsaad@cisco.com>

     Editor:   Himanshu Shah
               <mailto:hshah@ciena.com>

     Editor:   Oscar Gonzalez De Dios
               <mailto:oscar.gonzalezdedios@telefonica.com>";
```

```
  description
    "The model allows time scheduling parameters to be specified.";

  revision 2018-02-26 {
    description "Initial revision";
    reference "TBD";
  }

  /*
   * Typedefs
   */
  typedef operation {
    type enumeration {
      enum configure {
        description
        "Create the configuration data.";
      }
      enum deconfigure {
        description
        "Remove the configuration data.";
      }
      enum set {
        description
        "Set the specified configuration data.";
      }
      enum reset {
        description
        "Revert the specified configuration data back to the
         original value.";
      }
    }
    description "Operation type.";
  }

  /*
   * Groupings
   */

  grouping schedule-config-attributes {
    description
      "A group of attributes for a schedule.";

    leaf inclusive-exclusive {
      type enumeration {
        enum inclusive {
          description
            "The schedule element is inclusive, i.e., the schedule
             specifies the time at which the element is enabled.";
```

```
          }
          enum exclusive {
            description
              "The schedule element is exclusive. i.e., the schedule
               specifies the time at which the element is disabled.";
          }
        }
        default "inclusive";
        description
          "Whether the list item is inclusive or exclusive.";
      }
      leaf start {
        type yang:date-and-time;
        description "Start time.";
      }
      leaf schedule-duration {
        type string {
          pattern
            'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
        }
        description "Schedule duration in ISO 8601 format.";
      }
      leaf repeat-interval {
        type string {
          pattern
            'R\d*/P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?'
            + '(\d+S)?';
        }
        description "Repeat interval in ISO 8601 format.";
      }
      leaf operation {
        type operation;
        default "configure";
        description
          "Operation type.";
      }
      anydata data-value {
        description
          "The data value applied to the leaf data node
           specified by data-objects.
           The format of the data value depends on the value of the
           leaf operation defined above:
           configure:   data-value is the sub-tree added to the
                        target object;
           deconfigure: data-value is the child to be deleted from
                        the target object;
           set:         the target object MULST be a leaf, and
                        data-value is the new value to be set to
```

```
                        the target object;
             reset:     data-value is ignored.";
      }
    } // schedule-config-attributes

    grouping schedule-config-notification {
      description
        "A group of attributes for a schedule notification.";

      notification execution {
        description
          "Notification event for an execution performed on a target
           object.";
        leaf operation {
          type operation;
          mandatory true;
          description "Operation type.";
        }
        leaf datetime {
          type yang:date-and-time;
          description
            "The date and time when the execution was performed.";
        }
        anydata results {
          description
            "This chunk of data contains the results of the execution
             performed on the target object. The results are the same
             or equivalent to the contents of a <rpc-reply> message,
             Because of the nature of such a target execution, a
             <rpc-reply> message is not used to return the execution
             results. Instead, this notification is used to serve
             the same purpose.";
        }
      }
    } // schedule-config-notification

    grouping schedule-state-attributes {
      description
        "State attributes for a schedule.";
      container future-executions {
        description
          "The state information of the nexte scheduled event.";
        list execution {
          key "start";
          description
            "List of scheduled future executions.";
          leaf start {
            type yang:date-and-time;
```

```
            description "Start time.";
          }
          leaf duration {
            type string {
              pattern
                'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
            }
            description "Schedule duration in ISO 8601 format.";
          }
          leaf operation {
            type operation;
            description "Operation type.";
          }
        } // event
      } // future-events
    } // schedule-state-attributes

    grouping schedules {
      description
        "A list of schedules defining when a particular
         configuration takes effect.";
      container schedules {
        description
          "Container of a schedule list defining when a particular
           configuration takes effect.";
        list schedule {
          key "schedule-id";
          description "A list of schedule elements.";
          leaf schedule-id {
            type uint32;
            description "Identifies the schedule element.";
          }
          uses schedule-config-attributes;
        }
      }
    } // schedules

    /*
     * Configuration data and operational state nodes
     */
    container configuration-schedules {
      description
        "Serves as top-level container for a list of configuration
         schedules.";
      list target {
        key "object";
        description
          "A list of targets that configuration schedules are
```

```
           applied.";
         leaf object {
           type yang:xpath1.0;
           description
             "Xpath defining the data items of interest.";
         }
         uses schedules;
         container state {
           config false;
           description
             "Operational state data.";
           uses schedule-state-attributes;
         } // state

         uses schedule-config-notification;
       } // target
     } // configuration-schedules
   }
   <CODE ENDS>
```

8. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URI in the IETF XML registry [RFC3688]:

```
---------------------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-schedule
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
---------------------------------------------------------------------
```

This document registers the following YANG module in the YANG Module Names registry [RFC6020]:

```
---------------------------------------------------------------------
name:         ietf-schedule
namespace:    urn:ietf:params:xml:ns:yang:ietf-schedule
prefix:       l3te
reference:    RFC XXXX
---------------------------------------------------------------------
```

9. Security Considerations

The configuration, state, action and notification data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (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 contents.

The functionality defined in this memo can potentially allow network reconnaissance; by gathering information about schedules an attacker can learn about the network policy, its temporal behavior, and future events.

The schedule YANG model defines schedules that are writable, creatable, and deletable. Therefore, this model may be considered sensitive or vulnerable in some network environments. An attacker may maliciously configure a schedule in a way that disrupts the normal behavior of the network. Furthermore, an attacker may attempt to maliciously set a schedule or a set of schedules in a way that amplifies an attack, or schedules an attack to a particularly sensitive time instant.

The use of configuration scheduling implicitly assumes that there is an underlying synchronization or time distribution mechanism. Therefore, an attack on the synchronization mechanism may compromise the configuration scheduling. The security considerations of time protocols are discussed further in [RFC7384].

10. Contributors

Tal Mizrahi

Email: talmi@marvell.com

11. References

11.1. Normative References

[IEEE1588]
IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Version 2", IEEE Standard 1588.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

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

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <https://www.rfc-editor.org/info/rfc6021>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[RFC7399] Farrel, A. and D. King, "Unanswered Questions in the Path Computation Element Architecture", RFC 7399, DOI 10.17487/RFC7399, October 2014, <https://www.rfc-editor.org/info/rfc7399>.

[RFC7758] Mizrahi, T. and Y. Moses, "Time Capability in NETCONF", RFC 7758, DOI 10.17487/RFC7758, February 2016, <https://www.rfc-editor.org/info/rfc7758>.

[I-D.birrane-dtn-ama] Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-06 (work in progress), October 2017.

[I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-09 (work in progress), January 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-14 (work in progress), February 2018.

11.2. Informative References

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <https://www.rfc-editor.org/info/rfc6087>.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

EMail: Xufeng_Liu@jabil.com

Igor Bryskin
Huawei Technologies

EMail: Igor.Bryskin@huawei.com

Vishnu Pavan Beeram
Juniper Networks

EMail: vbeeram@juniper.net

Tarek Saad
Cisco Systems Inc

EMail: tsaad@cisco.com

Himanshu Shah
Ciena

EMail: hshah@ciena.com

Oscar Gonzalez de Dios
Telefonica

EMail: oscar.gonzalezdedios@telefonica.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

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction 2
2. Model catalog and registry requirements 3
3. Organizing YANG modules 5
3.1. Module information 5
4. Identifying interoperable models 7
4.1. Release bundle information 7
5. Specifying functionality with feature bundles 8
5.1. Feature bundle information 9
6. Module implementations 10
6.1. Implementation information 10
7. Security Considerations 10
8. IANA Considerations 11
9. YANG modules 11
10. References 38
10.1. Normative references 38
10.2. Informative references 38
Appendix A. Change summary 39
A.1. Changes between revisions -01 and -02 39
A.2. Changes between revisions -00 and -01 39
Authors' Addresses 39

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
Updates: rfc6087bis (if approved)
Intended status: Standards Track
Expires: August 12, 2017

C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
D. Bogdanovic

February 8, 2017

YANG Module Tags
draft-rtgyangdt-netmod-module-tags-00

Abstract

This document defines two modules that support the association of tags with modules. Tags may be included in a module or associated with a module through the use of an augmentation to YANG library that is defined in this document. The expectation is for such tags to be used to help classify and organize modules. Tags may be standardized and assigned during module definition; assigned by implementations; or dynamically defined and set by users. This document provides guidance to future model writers and, as such, this document updates [I-D.ietf-netmod-rfc6087bis].

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 August 12, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Conventions Used in This Document 3
3. Tag Locations 3
4. Tag Prefixes 3
4.1. IETF Standard Tags 4
4.2. Vendor Tags 4
4.3. Local Tags 4
4.4. Reserved Tags 4
5. Tag Management 4
5.1. Module Definition Association 4
5.2. Implementation Association 4
5.3. Administrative Tagging 4
5.3.1. Adding Tags 5
5.3.2. Removing Tags 5
5.3.3. Resetting Tags 5
6. Tags Module Structure 6
6.1. Tags Module Tree 6
6.2. Tags Module 6
7. Library Augmentation 8
7.1. Library Augmentation Module 9
8. Other Classifications 10
9. Guidelines to Model Writers 11
9.1. Include Module Tags 11
9.2. Define Standard Tags 12
10. IANA Considerations 12
10.1. YANG Module Tag Prefix Registry 12
10.2. YANG Module IETF Tag Registry 12
11. References 14
11.1. Normative References 14
11.2. Informative References 15
Authors' Addresses 15

1. Introduction

The use of tags for classification and organization is fairly ubiquitous not only within IETF protocols, but in the internet itself (see #hashtags). Tags can be usefully standardized, but they can also serve as a non-standardized mechanism available for users to define themselves. Our solution provides for both cases allowing for

the most flexibility. In particular, tags may be standardized and assigned during module definition; assigned by implementations; or dynamically defined and set by users.

This document defines two modules that support the association of tags with modules. The first module defines a grouping which contains a list of tags as well as rpc statements for changing the contents of the list. Tags are strings that are structured to enable the differentiation of globally assigned and non-assigned tags based on a fixed prefix. This document also defines an initial set of globally assigned tags.

The second module defined in this document defines an augmentation to YANG Library [RFC7895]. It uses (imports) the first module to provide a well known location for tags.

Section 9 provides guidelines for authors of YANG data models. This section updates [I-D.ietf-netmod-rfc6087bis].

2. Conventions Used in This Document

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

Note that lower case versions of these key words are used in section Section 9 where guidance is provided to future document authors.

3. Tag Locations

Two tag list locations are defined. One location is within the module itself, and the other location is in the yang library under the modules entry. When a module includes tags, the same tag list may also be presented in yang library.

To add tags to a module, the module definition includes a tag list using the ’module-tags’ grouping defined in this document. This list MUST be added by a module author under container named "module-tags" at the root of their module.

4. Tag Prefixes

All tags have a prefix indicating who owns their definition. An IANA registry is used to support standardizing tag prefixes. Currently 2 prefixes are defined with all others reserved.

4.1. IETF Standard Tags

An IETF standard tag is a tag that has the prefix "ietf:". All IETF standard tags are registered with IANA in a registry defined later in this document.

4.2. Vendor Tags

A vendor tag is a tag that has the prefix "vendor:". These tags are defined by the vendor that implements the module, and are not standardized.

4.3. Local Tags

A local tag is any tag that has the prefix "local:". These tags are defined by the local user/administrator, and will never be standardized.

4.4. Reserved Tags

Any tag not starting with the prefix "ietf:", "vendor:" or "local:" is reserved for future standardization.

5. Tag Management

Tags can become associated with a module in a number of ways. Tags may be defined as associated at model design time, at implementation time, or via user administrative control. As the main consumer of tags are users, users may remove any tag, no matter how the tag became associated with a module.

5.1. Module Definition Association

A module definition SHOULD indicate a set of standard tags to be automatically added by the module implementer. These tags MUST be standard tags (Section 4.1). This does imply that new modules may also drive the addition of new standard tags to the IANA registry.

5.2. Implementation Association

An implementation MAY include additional tags associated with a module. These tags may be standard or vendor specific tags.

5.3. Administrative Tagging

RPC statements are defined in this document to enable administrative addition and removal of tags from a module by a user. An additional

rpc is defined to reset a module's tag list to the implementation default.

Each rpc identifies the module on which the tag operation is to be performed. This identifications reuses the format of the common-leafs (sub) grouping defined in [RFC7895]. The grouping itself is refined in Section 6 so that it is a stand-alone grouping.

Implementations that support the rpc statements defined in this document MUST ensure that a specific module's tags leaf list is consistent across any location from which the list is available. Specifically this includes in the module itself, per Section 9.1, or in yang library, per Section 7.

Implementations that do not support the defined rpc statements (whether at all, or just for a particular rpc or module) MUST respond with an YANG transport protocol-appropriate rpc layer error when such a statement is received.

5.3.1. Adding Tags

The "add-tags" rpc statement is defined to support the addition of tags. This rpc statement takes as input module identification information and the list of tags to add.

No restriction is placed on the tag values to add.

5.3.2. Removing Tags

The "remove-tags" rpc statement is defined to remove tags. This rpc statement takes as input module identification information and the list of tags to remove.

No restriction is placed on the tag values to remove. This means that tags associated based on a module's definition or implementation MUST be removable.

5.3.3. Resetting Tags

The "reset-tags" rpc statement is defined to reset a module's tag list to the implementation default, i.e. the tags that are present based on module definition and any that are added during implementation time. This rpc statement takes module identification information as input, and provides the list of list of tags that are present after the reset.

6. Tags Module Structure

6.1. Tags Module Tree

The tree associated with the tags module is:

```
module: ietf-module-tags
  rpcs:
    +---x add-tags
    |  +---w input
    |     +---w name        yang:yang-identifier
    |     +---w revision?   union
    |     +---w tags*       string
    +---x remove-tags
    |  +---w input
    |     +---w name        yang:yang-identifier
    |     +---w revision?   union
    |     +---w tags*       string
    +---x reset-tags
       +---w input
       |  +---w name        yang:yang-identifier
       |  +---w revision?   union
       +--ro output
          +--ro tags*   string
```

6.2. Tags Module

```
<CODE BEGINS> file "ietf-module-tags@2017-02-08.yang"
module ietf-module-tags {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-module-tags";
  prefix "mtags";

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-yang-library {
    prefix yanglib;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
    "NetMod Working Group - <netmod@ietf.org>";

  description
```

```
      "This module describes a tagging mechanism for yang module.
       Tags may be IANA assigned or privately defined types.";

    revision "2017-02-08" {
      description
        "Initial revision.";
      reference "TBD";
    }

    grouping module-tags {
      description
        "A grouping that may be used to classify a module.";

      leaf-list tags {
        type string;

        config false;

        description
          "The module associated tags. See the IANA 'YANG Module Tag
          Prefix' registry for reserved prefixes and the IANA 'YANG
          Module IETF Tag' registry for IETF standard tags";
      }
    }

    grouping yanglib-common-leafs {
      description
        "Common parameters for YANG modules and submodules.
         This definition extract from RFC7895 as it is defined as
         a grouping within a grouping.

         TBD is there a legal way to use a grouping defined wuthin
         another grouping without using the parent? If so, should change
         to that.";

      leaf name {
        type yang:yang-identifier;
        mandatory true;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string { length 0; }
        }
        description
          "The YANG module or submodule revision date.
```

```
            A zero-length string is used if no revision statement
            is present in the YANG module or submodule.";
      }
    }

    rpc add-tags {
      description
        "Add a list of tags to a given module.";

      input {
        uses yanglib-common-leafs;
        uses module-tags;
      }
    }

    rpc remove-tags {
      description
        "Remove a list of tags, if present, from a given module.";

      input {
        uses yanglib-common-leafs;
        uses module-tags;
      }
    }

    rpc reset-tags {
      description
        "Reset a list of tags for a given module to the list of module
         and implementation time defiend tags. It provides the list of
         tags associated with the module post reset.";

      input {
        uses yanglib-common-leafs;
      }

      output {
         uses module-tags;
      }
    }
  }
  <CODE ENDS>
```

7. Library Augmentation

Tags can also be associated with a module using the yang library [RFC7895]. When a server supports both yang library and the augmentation defined below, a user can add, remove and search for tags for any module on the server regardless of whether the specific

module included tag support in its definition or not. If a server supports ietf-module-tags and the yang library it SHOULD also support the ietf-library-tags module.

The tree associated with the defined augmentation is:

```
module: ietf-library-tags
  augment /yanglib:modules-state/yanglib:module:
    +--ro tags*   string
```

7.1. Library Augmentation Module

```
<CODE BEGINS> file "ietf-library-tags@2017-02-08.yang"
module ietf-library-tags {
  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-library-tags";

  prefix ylibtags;

  import ietf-yang-library {
     prefix yanglib;
  }
  import ietf-module-tags {
     prefix mtags;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
      "NetMod Working Group - <netmod@ietf.org>";

  description
    "This module augments ietf-yang-library with searchable
    classfication tags.  Tags may be IANA or privately defined
    types.";

  revision "2017-02-06" {
    description
      "Initial revision.";
    reference "RFC TBD";
  }

  augment "/yanglib:modules-state/yanglib:module" {
    description
      "The yang library structure is augmented with a module tags
       list. This allows operators to tag modules regardless of
       whether the modules included tag support or not";

    uses mtags:module-tags;

  }
}
<CODE ENDS>
```

8. Other Classifications

It’s worth noting that a different yang module classification document exists [I-D.ietf-netmod-yang-model-classification]. That document is classifying modules in only a logical manner and does not

define tagging or any other mechanisms. It divides yang modules into 2 categories (service or entity) and then into one of 3 origins: standard, vendor or user. It does provide a good way to discuss and identify modules in general. This document defines standard tags to support [I-D.ietf-netmod-yang-model-classification] style classification.

9. Guidelines to Model Writers

This section updates [I-D.ietf-netmod-rfc6087bis]. This document makes two recommendations to model writers,

9.1. Include Module Tags

The correct way to use the module-tags grouping is to include it in a standard location at the top level of your module, specifically contained within a container named "module-tags". This standard location allows searching module using a well-known xpath wilcard path. For example:

```
module sample-module {
  ...
  import ietf-module-tags {
    prefix mtags;
  }
  ...
  container module-tags {
    description
      "A list of classification tags associated with this
       module. The following predefined tags <MUST|SHOULD|MAY>
       be included by an implementation:
         - ietf:foo
         - ietf:bar
         - ...
      ";
    uses mtags:module-tags;
  }
  ...
}
```

The associated tree will look like:

```
module: sample-module
    +--rw module-tags
    |  +--ro tags*   string
    +--...
```

9.2. Define Standard Tags

A module should indicate, in the description of the "module-tags" container, the set of tags that are to be populated in the leaf-list for any implementation of the module. This description should also include the appropriate conformance statement or statements, using [RFC2119] language, for each tag.

The module writer may use existing standard tags, or use new tags defined in the model definition, as appropriate. New tags should be assigned in the IANA registry defined below, see Section 10.2 below.

10. IANA Considerations

10.1. YANG Module Tag Prefix Registry

This registry allocates tag prefixes. All YANG module tags must begin with one of the prefixes in this registry.

The allocation policy for this registry is Specification Required [RFC5226].

The initial values for this registry are as follows.

prefix	description
ietf:	IETF Standard Tag allocated in the IANA YANG Module IETF Tag Registry.
vendor:	Non-standardized tags allocated by the module implementer.
local:	Non-standardized tags allocated by and for the user.

10.2. YANG Module IETF Tag Registry

This registry allocates prefixes that have the standard prefix "ietf:". New values should be well considered and not achievable through a combination of already existing standard tags.

The allocation policy for this registry is IETF Review [RFC5226].

The initial values for this registry are as follows.

[Editor's note: some of these tags are expected to move to [I-D.ietf-rtgwg-device-model] if/when this document becomes a WG document and that document is refactored to use tags.]

Tag	Description	Reference
ietf:area:art	Applications and Real-Time Area module.	[This document]
ietf:area:gen	General Area module.	[This document]
ietf:area:int	Internet Area module.	[This document]
ietf:area:ops	Operations and Management Area module.	[This document]
ietf:area:rtg	Routing Area module.	[This document]
ietf:area:sec	Security Area module.	[This document]
ietf:area:tsv	Transport Area module.	[This document]
ietf:entity	A module for an entity (*).	[This document]
ietf:service	A module for a service (*).	[This document]
ietf:hardware	A module for hardware.	[This document]
ietf:software	A module for software.	[This document]
ietf:protocol	A module representing a protocol.	[This document]
ietf:protocol:system-management	A module representing a	[This document]

	system management protocol.	
ietf:protocol:network-service	A module representing a network service protocol.	[This document]
ietf:protocol:routing	A module representing a control plane routing protocol.	[This document]
ietf:protocol:signaling	A module representing a control plane signaling protocol.	[This document]
ietf:protocol:oam	A module representing a Operations, Administration, and Maintenance protocol.	[This document]
ietf:protocol:lmp	A module representing a link management protocol.	[This document]
ietf:protocol:routing:igp	An IGP protocol module.	[This document]
ietf:protocol:routing:egp	An EGP protocol module.	[This document]

(*) - see [I-D.ietf-netmod-yang-model-classification]

Table 1: IETF Module Tag Registry

11. References

11.1. Normative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-10 (work in progress), January 2017.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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>.

11.2. Informative References

[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.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-ietf-rtgwg-device-model-01 (work in progress), October 2016.

Authors' Addresses

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

Network Working Group
Internet-Draft
Updates: rfc6087bis (if approved)
Intended status: Standards Track
Expires: April 27, 2018

C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
D. Bogdanovic

October 24, 2017

YANG Module Tags
draft-rtgyangdt-netmod-module-tags-02

Abstract

This document provides for the association of tags with YANG modules. The expectation is for such tags to be used to help classify and organize modules. A method for defining, reading and writing a modules tags is provided, as well as an augmentation to YANG library. Tags may be standardized and assigned during module definition; assigned by implementations; or dynamically defined and set by users. This document provides guidance to future model writers and, as such, this document updates [I-D.ietf-netmod-rfc6087bis].

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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on April 27, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Conventions Used in This Document 3
3. Tag Locations 3
4. Tag Prefixes 3
4.1. IETF Standard Tags 3
4.2. Vendor Tags 3
4.3. Local Tags 4
4.4. Reserved Tags 4
5. Tag Management 4
5.1. Module Definition Association 4
5.2. Implementation Association 4
5.3. Administrative Tagging 4
5.3.1. Resetting Tags 5
6. Tags Module Structure 5
6.1. Tags Module Tree 5
6.2. Tags Module 5
7. Library Augmentation 7
7.1. Library Augmentation Module 8
8. Other Classifications 9
9. Guidelines to Model Writers 9
9.1. Define Standard Tags 9
10. IANA Considerations 10
10.1. YANG Module Tag Prefix Registry 10
10.2. YANG Module IETF Tag Registry 10
11. References 12
11.1. Normative References 12
11.2. Informative References 12
Authors' Addresses 12

1. Introduction

The use of tags for classification and organization is fairly ubiquitous not only within IETF protocols, but in the internet itself (e.g., #hashtags). Tags can be usefully standardized, but they can also serve as a non-standardized mechanism available for users to define themselves. Our solution provides for both cases allowing for the most flexibility. In particular, tags may be standardized as well as assigned during module definition; assigned by implementations; or dynamically defined and set by users.

This document defines two modules. The first module defines a list of module entries to allow for adding or removing of tags. It also defines an RPC to reset a modules tags to the original values. The second module defines an augmentation to YANG Library [RFC7895] to allow for reading a modules tags.

This document also defines an IANA registry for tag prefixes as well as a set of globally assigned tags.

Section 9 provides guidelines for authors of YANG data models. This section updates [I-D.ietf-netmod-rfc6087bis].

2. Conventions Used in This Document

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

Note that lower case versions of these key words are used in section Section 9 where guidance is provided to future document authors.

3. Tag Locations

Each module has only one logical tag list; however, that tag list may be accessed from multiple locations.

We define two tag list locations. The first location is used for configuration and is a top level list of module entries where each entry contain the list of tags. The second read-only location is through the yang library under the module entry.

4. Tag Prefixes

All tags have a prefix indicating who owns their definition. An IANA registry is used to support standardizing tag prefixes. Currently 3 prefixes are defined with all others reserved.

4.1. IETF Standard Tags

An IETF standard tag is a tag that has the prefix "ietf:". All IETF standard tags are registered with IANA in a registry defined later in this document.

4.2. Vendor Tags

A vendor tag is a tag that has the prefix "vendor:". These tags are defined by the vendor that implements the module, and are not standardized; however, it is recommended that the vendor consider

including extra identification in the tag name to avoid collisions (e.g., vendor:super-duper-company:...).

4.3. Local Tags

A local tag is any tag that has the prefix "local:". These tags are defined by the local user/administrator and will never be standardized.

4.4. Reserved Tags

Any tag not starting with the prefix "ietf:", "vendor:" or "local:" is reserved for future standardization.

5. Tag Management

Tags can become associated with a module in a number of ways. Tags may be defined and associated at model design time, at implementation time, or via user administrative control. As the main consumer of tags are users, users may also remove any tag, no matter how the tag became associated with a module.

5.1. Module Definition Association

A module definition SHOULD indicate a set of tags to be automatically added by the module implementer. These tags MUST be standard tags (Section 4.1). This does imply that new modules may also drive the addition of new standard tags to the IANA registry.

5.2. Implementation Association

An implementation MAY include additional tags associated with a module. These tags may be standard or vendor specific tags.

5.3. Administrative Tagging

Tags of any kind can be assigned and removed with normal configuration mechanisms. Additionally we define an RPC to reset a module’s tag list to the implementation default.

Implementations MUST ensure that a modules tag list is consistent across any location from which the list is accessible. So if a user adds a tag through configuration that tag should also be seen when using the yang library augmentation.

Implementations that do not support the reset rpc statement (whether at all, or just for a particular rpc or module) MUST respond with an

YANG transport protocol-appropriate rpc layer error when such a statement is received.

5.3.1. Resetting Tags

The "reset-tags" rpc statement is defined to reset a module's tag list to the implementation default, i.e. the tags that are present based on module definition and any that are added during implementation time. This rpc statement takes module identification information as input, and provides the list of tags that are present after the reset.

6. Tags Module Structure

6.1. Tags Module Tree

The tree associated with the tags module is:

```
module: ietf-module-tags
  rpcs:
    +---x reset-tags
       +---w input
       |  +---w name        yang:yang-identifier
       |  +---w revision?   union
       +--ro output
          +--ro tags*   string
```

6.2. Tags Module

```
<CODE BEGINS> file "ietf-module-tags@2017-10-25.yang"
module ietf-module-tags {
  yang-version "1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-module-tags";
  prefix "mtags";

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-yang-library {
    prefix yanglib;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
    "NetMod Working Group - <netmod@ietf.org>";
```

```
    description
      "This module describes a tagging mechanism for yang module.
       Tags may be IANA assigned or privately defined types.";

    revision "2017-10-25" {
      description
        "Initial revision.";
      reference "TBD";
    }

    grouping module-tags {
      description
        "A grouping that may be used to classify a module.";

      leaf-list tags {
        type string;

        description
          "The module associated tags. See the IANA 'YANG Module Tag
          Prefix' registry for reserved prefixes and the IANA 'YANG
          Module IETF Tag' registry for IETF standard tags";
      }
    }

    grouping yanglib-common-leafs {
      description
        "Common parameters for YANG modules and submodules.
         This definition extract from RFC7895 as it is defined as
         a grouping within a grouping.

         TBD is there a legal way to use a grouping defined within
         another grouping without using the parent? If so, should change
         to that.";

      leaf name {
        type yang:yang-identifier;
        mandatory true;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string { length 0; }
        }
        mandatory true;
        description
          "The YANG module or submodule revision date.
```

```
            A zero-length string is used if no revision statement
            is present in the YANG module or submodule.";
      }
    }

    list module-tags {
      key "name revision";
      description
        "A list of modules and their tags";
      uses yanglib-common-leafs; // uses yanglib:common-leafs;
      uses module-tags;
    }

    rpc reset-tags {
      description
        "Reset a list of tags for a given module to the list of module
         and implementation time defiend tags. It provides the list of
         tags associated with the module post reset.";

      input {
        uses yanglib-common-leafs; // uses yanglib:common-leafs;
      }

      output {
         uses module-tags;
      }
    }
  }
  <CODE ENDS>
```

7. Library Augmentation

A modules tags can also be read using the yang library [RFC7895] if a server supports both YANG library and the augmentation defined below. If a server supports ietf-module-tags and the YANG library it SHOULD also support the ietf-library-tags module.

The tree associated with the defined augmentation is:

```
module: ietf-library-tags
  augment /yanglib:modules-state/yanglib:module:
    +--ro tags*   string
```

7.1. Library Augmentation Module

```
<CODE BEGINS> file "ietf-library-tags@2017-08-12.yang"
module ietf-library-tags {
  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-library-tags";

  prefix ylibtags;

  import ietf-yang-library {
     prefix yanglib;
  }
  import ietf-module-tags {
     prefix mtags;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
      "NetMod Working Group - <netmod@ietf.org>";

  description
    "This module augments ietf-yang-library with searchable
    classfication tags.  Tags may be IANA or privately defined
    types.";

  revision "2017-08-12" {
    description
      "Initial revision.";
    reference "RFC TBD";
  }

  augment "/yanglib:modules-state/yanglib:module" {
    description
      "The yang library structure is augmented with a module tags
       list. This allows operators to tag modules regardless of
       whether the modules included tag support or not";

    uses mtags:module-tags;

  }
}
<CODE ENDS>
```

8. Other Classifications

It's worth noting that a different yang module classification document exists [RFC8199]. That document is classifying modules in only a logical manner and does not define tagging or any other mechanisms. It divides yang modules into 2 categories (service or element) and then into one of 3 origins: standard, vendor or user. It does provide a good way to discuss and identify modules in general. This document defines standard tags to support [RFC8199] style classification.

9. Guidelines to Model Writers

This section updates [I-D.ietf-netmod-rfc6087bis].

9.1. Define Standard Tags

A module SHOULD indicate, in the description statement of the module, a set of tags that are to be associated with it. This description should also include the appropriate conformance statement or statements, using [RFC2119] language for each tag.

```
      module sample-module {
        ...
        description
          "[Text describing the module...]

          RFC<this document> TAGS:
          The following tags MUST be included by an implementation:
               - ietf:some-required-tag:foo
               - ...
          The following tags SHOULD be included by an implementation:
               - ietf:some-recommended-tag:bar
               - ...
          The following tags MAY be included by an implementation:
               - ietf:some-optional-tag:baz
               - ...
            ";
        ...
      }
```

One SHOULD only include conformance text if there will be tags listed (i.e., there's no need to indicate an empty set).

The module writer may use existing standard tags, or use new tags defined in the model definition, as appropriate. New tags should be assigned in the IANA registry defined below, see Section 10.2 below.

10. IANA Considerations

10.1. YANG Module Tag Prefix Registry

This registry allocates tag prefixes. All YANG module tags SHOULD begin with one of the prefixes in this registry.

The allocation policy for this registry is Specification Required [RFC5226].

The initial values for this registry are as follows.

prefix	description
ietf:	IETF Standard Tag allocated in the IANA YANG Module IETF Tag Registry.
vendor:	Non-standardized tags allocated by the module implementer.
local:	Non-standardized tags allocated by and for the user.

Other SDOs (standard organizations) wishing to standardize their own set of tags could allocate a top level prefix from this registry.

10.2. YANG Module IETF Tag Registry

This registry allocates prefixes that have the standard prefix "ietf:". New values should be well considered and not achievable through a combination of already existing standard tags.

The allocation policy for this registry is IETF Review [RFC5226].

The initial values for this registry are as follows.

[Editor’s note: many of these tags may move to [I-D.ietf-rtgwg-device-model] if/when that document is refactored to use tags.]

Tag	Description	Reference
ietf:rfc8199:element	A module for a network element.	[RFC8199]
ietf:rfc8199:service	A module for a network service.	[RFC8199]
ietf:rfc8199:standard	A module defined by a standards organization.	[RFC8199]

ietf:rfc8199:vendor	A module defined by a vendor.	[RFC8199]
ietf:rfc8199:user	A module defined by the user.	[RFC8199]
ietf:device:hardware	A module relating to device hardware (e.g., inventory).	[This document]
ietf:device:software	A module relating to device software (e.g., installed OS).	[This document]
ietf:device:qos	A module for managing quality of service.	[This document]
ietf:protocol	A module representing a protocol.	[This document]
ietf:system-management	A module relating to system management (e.g., a system management protocol).	[This document]
ietf:network-service	A module relating to network service (e.g., a network service protocol).	[This document]
ietf:oam	A module representing Operations, Administration, and Maintenance.	[This document]
ietf:routing	A module related to routing.	[This document]
ietf:routing:rib	A module related to routing information bases.	[This document]
ietf:routing:igp	An interior gateway protocol module.	[This document]
ietf:routing:egp	An exterior gateway protocol module.	[This document]
ietf:signaling	A module representing control plane signaling.	[This document]
ietf:lmp	A module representing a link management protocol.	[This document]

+-----------------------+------------------------------+-----------+

Table 1: IETF Module Tag Registry

11. References

11.1. Normative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-14 (work in progress), September 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

11.2. Informative References

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

Authors’ Addresses

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com