

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2019

Q. Wu
Huawei
B. Lengyel
Ericsson Hungary
Y. Niu
Huawei
June 29, 2019

Factory Default Setting
draft-ietf-netmod-factory-default-02

Abstract

This document defines a method to reset a server to its factory-default content. The reset operation may be used e.g. during initial zero-touch configuration or when the existing configuration has major errors, so re-starting the configuration process from scratch is the best option.

A new factory-reset RPC is defined. Several methods of documenting the factory-default content are specified.

Optionally a new "factory-default" read-only datastore is defined, that contains the data that will be copied over to the running datastore at reset.

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 December 31, 2019.

Copyright Notice

Copyright (c) 2019 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
1.1. Terminology	3
2. Factory-Reset RPC	4
3. Factory-Default Datastore	4
4. YANG Module	5
5. IANA Considerations	7
6. Security Considerations	7
7. Acknowledgements	7
8. Contributors	8
9. References	8
9.1. Normative References	8
9.2. Informative References	8
Appendix A. Open Issues	9
Appendix B. Difference between <startup> datastore and <factory- default> datastore	9
Appendix C. Changes between revisions	9
Authors' Addresses	10

1. Introduction

This document defines a method to reset a server to its factory-default content. The reset operation may be used e.g. during initial zero-touch configuration [RFC8572] or when the existing configuration has major errors, so re-starting the configuration process from scratch is the best option. When resetting a datastore all previous configuration settings will be lost and replaced by the factory-default content.

A new factory-reset RPC is defined. Several methods of documenting the factory-default content are specified.

Optionally a new "factory-default" read-only datastore is defined, that contains the data that will be copied over to all read-write configuration datastores at reset. This datastore can also be used in <get-data> or <get-config> operations.

NETCONF defines the <delete> operation that allows resetting the <startup> datastore and the <discard-changes> operation that copies the content of the <running> datastore into the <candidate> datastore. However it is not possible to reset the running datastore, to reset the candidate datastore without changing the running datastore or to reset any dynamic datastore.

A RESTCONF server MAY implement the above NETCONF operations, but that would still not allow it to reset the running configuration.

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC8342] and are not redefined here:

- o server
- o startup configuration datastore
- o candidate configuration datastore
- o running configuration datastore
- o intended configuration datastore
- o operational state datastore

The following terms are defined in this document as follows:

- o factory-default datastore: A read-only datastore holding a preconfigured minimal initial configuration that can be used to initialize the configuration of a server. The content of the datastore is usually static, but MAY depend on external factors like available HW.

2. Factory-Reset RPC

A new "factory-reset" RPC is introduced. Upon receiving the RPC the server resets the content of all read-write configuration datastores (e.g., <running> and <startup>) to its factory-default content. Read-only datastores receive their content from other datastores (e.g. <intended> gets its content from <running>).

Factory-default content SHALL be specified by one of the following means in order of precedence

1. For the <running>, <candidate> and <startup> datastores as the content of the <factory-default> datastore, if it exists
2. YANG Instance Data [I-D.ietf-netmod-yang-instance-file-format]
3. In some implementation specific manner
4. For dynamic datastores unless otherwise specified the factory-default content is empty.

In addition to set the content of the read-write configuration datastores, the "factory-reset" RPC might also be used to clean up files, restart the node or some of the SW processes, or it might set some security data/passwords to the default value, remove logs, remove any temporary data (from datastore or elsewhere) etc.

3. Factory-Default Datastore

Following guidelines for defining Datastores in the appendix A of [RFC8342], this document introduces a new datastore resource named 'Factory-Default' that represents a preconfigured minimal initial configuration that can be used to initialize the configuration of a server.

- o Name: "factory-default"
- o YANG modules: all
- o YANG nodes: all "config true" data nodes
- o Management operations: The content of the datastore is set by the server in an implementation dependent manner. The content can not be changed by management operations via NETCONF, RESTCONF, the CLI etc. unless specialized, dedicated operations are provided. The contents of the datastore can be read using NETCONF, RESTCONF <get-data> and <get-config> operations. The operation <factory-reset> can be used to copy the factory default content to a set of

read-write configuration datastores and then the content of these datastores is propagated automatically to any other read only datastores, e.g., <intended> and <operational>.

- o Origin: This document does not define a new origin identity as it does not interact with <operational> datastore.
- o Protocols: RESTCONF, NETCONF and other management protocol.
- o Defining YANG module: "ietf-factory-default".

The datastore content is usually defined by the device vendor. It is usually static, but MAY change e.g., depending on external factors like HW available or during device upgrade.

On devices that support non-volatile storage, the contents of <factory > MUST persist across restarts.

4. YANG Module

```
<CODE BEGINS> file "ietf-factory-default.yang"
module ietf-factory-default {
  yang-version 1.1;
  namespace urn:ietf:params:xml:ns:yang:ietf-factory-default;
  prefix fd;

  import ietf-netconf { prefix nc ; }
  import ietf-datastores { prefix ds; }

  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG Web:  <https://tools.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>

    Editor:   Balazs Lengyel
              <mailto:balazs.lengyel@ericsson.com>
    Editor:   Qin Wu
              <mailto:bill.wu@huawei.com>
    Editor:   Ye Niu
              <mailto:niuye@huawei.com>";
  description
    "This module defines the
    - factory-reset RPC
    - factory-default datastore

    It provides functionality to reset a server to its
    factory-default content.
```

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) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2019 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

```
revision 2019-05-03 {
  description
    "Initial revision.";
  reference "RFC XXXX: Factory default Setting";
}

feature factory-default-as-datastore {
  description "Indicates that the factory default configuration is
    also available as a separate datastore";
}

rpc factory-reset {
  description "The server resets the content of all read-write
    configuration datastores (e.g., <running> and <startup>) to
    its factory default content.";
}

identity factory-default {
  base ds:datastore;
  if-feature factory-default-as-datastore;
  description "The read-only datastore contains the configuration that
    will be copied into e.g., the running datastore by the
    factory-reset operation if the target is the running
    datastore.";
}

augment /nc:get-config/nc:input/nc:source/nc:config-source {
  if-feature factory-default-as-datastore;
  description "Allows the get-config operation to use the
    factory-default datastore as a source";
}
```

```
    leaf factory-default {  
        type empty ;  
        description  
            "The factory-default datastore is the source.";    }  
    }  
<CODE ENDS>
```

5. IANA Considerations

This document registers one URI in the IETF XML Registry [RFC3688]. The following registration has been made:

URI: urn:ietf:params:xml:ns:yang:ietf-factory-default

Registrant Contact: The IESG.

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

This document registers one YANG module in the YANG Module Names Registry [RFC6020]. The following registration has been made:

name: ietf-factory-default

namespace: urn:ietf:params:xml:ns:yang:ietf-factory-default

prefix: fd

RFC: xxxx

6. Security Considerations

The <factory-reset> RPC can overwrite important and security sensitive information in one of the other datastores e.g. running, therefore it is important to restrict access to this RPC using the standard access control methods. [RFC8341]

The content of the factory-default datastore is usually not security sensitive as it is the same on any device of a certain type. In case there is any sensitive content in the factory-default datastore, it should be protected in a secure way, e.g., sign or encrypt the sensitive information.

7. Acknowledgements

Thanks to Juergen Schoenwaelder, Ladislav Lhotka, Alex Campbell, Joe Clark, Robert Wilton, Kent Watsen, Joel Jaeggli, Andy Berman, Susan Hares to review this draft and provide important input to this document.

8. Contributors

Rohit R Ranade
Huawei
Email: rohitrranade@huawei.com

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, <<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>>.
- [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>>.

9.2. Informative References

- [I-D.ietf-netmod-yang-instance-file-format] Lengyel, B. and B. Claise, "YANG Instance Data File Format", draft-ietf-netmod-yang-instance-file-format-02 (work in progress), February 2019.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

Appendix A. Open Issues

- o Do we need an extra parameter that may order a restart of the YANG-server or the whole system?
- o Do we allow different datastore have different factory default content? No

Appendix B. Difference between <startup> datastore and <factory-default> datastore

When the device first boots up, the content of the <startup> and <factory-default> will be identical. The content of <startup> can be subsequently changed by using <startup> as a target in a <copy-config> operation. The <factory-default> is a read-only datastore and it is usually static as described in earlier sections.

Appendix C. Changes between revisions

v01 - v02

- o Address security issue in the security consideration section.
- o Remove an extension to the NETCONF <copy-config> operation which allows it to operate on the factory-default datastore.
- o Add an extension to the NETCONF <get-config> operation which allows it to operate on the factory-default datastore.

v00 - v01

- o Change YANG server into server defined in NMDA architecture based on discussion.
- o Allow reset the content of all read-write configuration datastores to its factory-default content except <candidate>.
- o Add clarification text on factory-reset protocol operation behavior.

v03 - v00

- o Change draft name from draft-wu to draft-ietf-netmod-factory-default-00 without content changes.

v02 - v03

- o Change reset-datastore RPC into factory-reset RPC to allow reset the whole device with factory default content.
- o Remove target datastore parameter from factory-reset RPC.
- o Other editorial changes.

v01 - v02

- o Add copy-config based on Rob's comment.
- o Reference Update.

v03 - v00 - v01

- o Changed name from draft-wu-netconf-restconf-factory-restore to draft-wu-netmod-factory-default
- o Removed copy-config ; reset-datastore is enough

v02 - v03

- o Restructured
- o Made new datastore optional
- o Removed Netconf capability
- o Listed Open issues

v01 - v02

- o -

v00 - v01

- o -

Authors' Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Balazs Lengyel
Ericsson Hungary
Magyar Tudosok korutja 11
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Ye Niu
Huawei

Email: niuye@huawei.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2019

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
March 5, 2019

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-07

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.

The YANG data model in this document conforms to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

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 September 6, 2019.

Copyright Notice

Copyright (c) 2019 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	5
3.2. Dampening	6
3.2.1. Suppress Threshold	7
3.2.2. Half-Life Period	7
3.2.3. Reuse Threshold	7
3.2.4. Maximum Suppress Time	7
3.3. Encapsulation	8
3.4. Loopback	8
3.5. Layer 2 MTU	8
3.6. Sub-interface	9
3.7. Forwarding Mode	9
4. Interfaces Ethernet-Like Module	10
5. Interfaces Common YANG Module	10
6. Interfaces Ethernet-Like YANG Module	21
7. Examples	24
7.1. Carrier delay configuration	24
7.2. Dampening configuration	25
7.3. MAC address configuration	26
8. Acknowledgements	27
9. ChangeLog	28
9.1. Version -07	28
9.2. Version -06	28
9.3. Version -05	28
9.4. Version -04	28
9.5. Version -03	28
9.6. Version -02	28
10. IANA Considerations	28
11. Security Considerations	28
11.1. interfaces-common.yang	29
11.2. interfaces-ethernet-like.yang	30
12. References	30
12.1. Normative References	30
12.2. Informative References	31
Authors' Addresses	31

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

Tree diagrams used in this document follow the notation defined in [RFC8340].

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:

- o A carrier delay feature used to provide control over short lived link state flaps.
- o An interface link state dampening feature that is used to provide control over longer lived link state flaps.
- o An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.
- o A loopback configuration leaf that is primarily aimed at loopback at the physical layer.

- o MTU configuration leaves applicable to all packet/frame based interfaces.
- o A forwarding mode leaf to indicate the OSI layer at which the interface handles traffic
- o 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 principle 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 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 augment 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:

- o Internal loopback - All egress traffic on the interface is internally looped back within the interface to be received on the ingress path.
- o Line loopback - All ingress traffic received on the interface is internally looped back within the interface to the egress path.
- o 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 12-mtu excludes the 4-8 byte overhead of any known (e.g. explicitly matched by a child sub-interface) 802.1Q 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:

- o Optical Layer - Traffic is being forwarded at the optical layer. This includes DWDM or OTN based switching.
- o Layer 2 - Layer 2 based forwarding, such as Ethernet/VLAN based switching, or L2VPN services.
- o 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@2019-03-05.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 8343) with common configurable layer 2
    properties.

    Copyright (c) 2016-2019 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 2019-03-05 {
  description
    "Initial version";

  reference "Internet draft: draft-ietf-netmod-intf-ext-yang-07";
}

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 capabilities 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) 802.1Q 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@2019-03-05.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: 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 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) 2016-2019 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 2019-03-05 {
 description "Initial revision";

reference
 "Internet draft: draft-ietf-netmod-intf-ext-yang-07";

}

/*

* 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 msec 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>50</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 -07

- o Minor editorial updates

9.2. Version -06

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

9.3. Version -05

- o Incorporate feedback from Andy Bierman

9.4. Version -04

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

9.5. Version -03

- o Fixed incorrect module name references, and updated tree output

9.6. Version -02

- o 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 two 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:

- o /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:

- o /if:interfaces/if:interface/carrier-delay/down
- o /if:interfaces/if:interface/carrier-delay/up
- o /if:interfaces/if:interface/dampening/half-life
- o /if:interfaces/if:interface/dampening/reuse
- o /if:interfaces/if:interface/dampening/suppress
- o /if:interfaces/if:interface/dampening/max-suppress-time

The following leaves could cause traffic loss on the interface because the received or transmitted frames do not comply with the frame matching criteria on the interface and hence would be dropped:

- o /if:interfaces/if:interface/encapsulation
- o /if:interfaces/if:interface/l2-mtu

- o /if:interfaces/if:interface/forwarding-mode

Normally devices will not allow the parent-interface leaf to be changed after the interface has been created. If an implementation did allow the parent-interface leaf to be changed then it could cause all traffic on the affected interface to be dropped. The affected leaf is:

- o /if:interfaces/if:interface/parent-interface

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. The module currently only contains a node for configuring the operational MAC address to use on an interface. Adding/modifying/deleting this leaf has the potential risk of causing protocol instability, excessive protocol traffic, and general traffic loss, particularly if the configuration change caused a duplicate MAC address to be present on the local network. The following leaf is affected:

- o interfaces/interface/ethernet-like/mac-address

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-04 (work in progress), July 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>>.
- [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>>.
- [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>>.

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: January 9, 2020

A. Clemm
Y. Qu
Futurewei
J. Tantsura
Apstra
A. Bierman
YumaWorks
July 8, 2019

Comparison of NMDA datastores
draft-ietf-netmod-nmda-diff-02

Abstract

This document defines an RPC operation to compare management datastores that comply with the NMDA architecture.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 9, 2020.

Copyright Notice

Copyright (c) 2019 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. Key Words	3
3. Definitions and Acronyms	3
4. Data Model Overview	4
5. YANG Data Model	5
6. Example	9
7. Open Issues	12
8. Possible Future Extensions	13
9. IANA Considerations	13
9.1. Updates to the IETF XML Registry	13
9.2. Updates to the YANG Module Names Registry	14
10. Security Considerations	14
11. Acknowledgments	15
12. References	15
12.1. Normative References	15
12.2. Informative References	16
Authors' Addresses	16

1. Introduction

The revised Network Management Datastore Architecture (NMDA) [RFC8342] introduces a set of new datastores that each hold YANG-defined data [RFC7950] and represent a different "viewpoint" on the data that is maintained by a server. New YANG datastores that are introduced include <intended>, which contains validated configuration data that a client application intends to be in effect, and <operational>, which contains at least conceptually operational state data (such as statistics) as well as configuration data that is actually in effect.

NMDA introduces in effect a concept of "lifecycle" for management data, allowing to clearly distinguish between data that is part of a configuration that was supplied by a user, configuration data that has actually been successfully applied and that is part of the operational state, and overall operational state that includes both applied configuration data as well as status and statistics.

As a result, data from the same management model can be reflected in multiple datastores. Clients need to specify the target datastore to be specific about which viewpoint of the data they want to access. This way, an application can differentiate whether they are (for example) interested in the configuration that has been applied and is

actually in effect, or in the configuration that was supplied by a client and that is supposed to be in effect.

Due to the fact that data can propagate from one datastore to another, it is possible for differences between datastores to occur. Some of this is entirely expected, as there may be a time lag between when a configuration is given to the device and reflected in <intended>, until when it actually takes effect and is reflected in <operational>. However, there may be cases when a configuration item that was to be applied may not actually take effect at all or needs an unusually long time to do so. This can be the case due to certain conditions not being met, resource dependencies not being resolved, or even implementation errors in corner conditions.

When configuration that is in effect is different from configuration that was applied, many issues can result. It becomes more difficult to operate the network properly due to limited visibility of actual status which makes it more difficult to analyze and understand what is going on in the network. Services may be negatively affected (for example, breaking a service instance resulting in service is not properly delivered to a customer) and network resources be misallocated.

Applications can potentially analyze any differences between two datastores by retrieving the contents from both datastores and comparing them. However, in many cases this will be at the same time costly and extremely wasteful.

This document introduces a YANG data model which defines RPCs, intended to be used in conjunction with NETCONF [RFC6241] or RESTCONF [RFC8040], that allow a client to request a server to compare two NMDA datastores and report any differences.

2. Key Words

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.

3. Definitions and Acronyms

NMDA: Network Management Datastore Architecture

RPC: Remote Procedure Call

4. Data Model Overview

At the core of the solution is a new management operation, <compare>, that allows to compare two datastores for the same data. The operation checks whether there are any differences in values or in data nodes that are contained in either datastore, and returns any differences as output. The output is returned in the format specified in YANG-Patch [RFC8072].

The YANG data model defines the <compare> operation as a new RPC. The operation takes the following input parameters:

- o source: The source identifies the datastore that will serve as reference for the comparison, for example <intended>.
- o target: The target identifies the datastore to compare against the source.
- o filter-spec: This is a choice between different filter constructs to identify the portions of the datastore to be retrieved. It acts as a node selector that specifies which data nodes are within the scope of the comparison and which nodes are outside the scope. This allows a comparison operation to be applied only to a specific portion of the datastore that is of interest, such as a particular subtree. (The filter does not contain expressions that would match values data nodes, as this is not required by most use cases and would complicate the scheme, from implementation to dealing with race conditions.)
- o all: When set, this parameter indicates that all differences should be included, including differences pertaining to schema nodes that exist in only one of the datastores. When this parameter is not included, a prefiltering step is automatically applied to exclude data from the comparison that does not pertain to both datastores: if the same schema node is not present in both datastores, then all instances of that schema node and all its descendants are excluded from the comparison. This allows client applications to focus on the differences that constitute true mismatches of instance data without needing to specify more complex filter constructs.

The operation provides the following output parameter:

- o differences: This parameter contains the list of differences. Those differences are encoded per YANG-Patch data model defined in RFC8072. The YANG-Patch data model is augmented to indicate the value of source datastore nodes in addition to the patch itself that would need to be applied to the source to produce the target.

When the target datastore is <operational>, "origin" metadata is included as part of the patch. Including origin metadata can help in some cases explain the cause of a difference, for example when a data node is part of <intended> but the origin of the same data node in <operational> is reported as "system".

The data model is defined in the ietf-nmda-compare YANG module. Its structure is shown in the following figure. The notation syntax follows [RFC8340].

```

module: ietf-nmda-compare
rpcs:
  +---x compare
    +---w input
      +---w source          identityref
      +---w target          identityref
      +---w all?             empty
      +---w (filter-spec)?
        +---:(subtree-filter)
          | +---w subtree-filter?
          +---:(xpath-filter)
            +---w xpath-filter?      yang:xpath1.0 {nc:xpath}?
    +---ro output
      +---ro (compare-response)?
        +---:(no-matches)
          | +---ro no-matches?      empty
          +---:(differences)
            +---ro differences
              +---ro yang-patch
                +---ro patch-id      string
                +---ro comment?      string
                +---ro edit* [edit-id]
                  +---ro edit-id      string
                  +---ro operation    enumeration
                  +---ro target       target-resource-offset
                  +---ro point?       target-resource-offset
                  +---ro where?       enumeration
                  +---ro value?
                  +---ro source-value?

```

Structure of ietf-nmda-compare

5. YANG Data Model

```

<CODE BEGINS> file "ietf-nmda-compare@2019-07-08.yang"
module ietf-nmda-compare {

  yang-version 1.1;

```



```
namespace "urn:ietf:params:xml:ns:yang:ietf-nmda-compare";

prefix cp;

import ietf-yang-types {
  prefix yang;
}
import ietf-datastores {
  prefix ds;
}
import ietf-yang-patch {
  prefix ypatch;
}
import ietf-netconf {
  prefix nc;
}

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

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

  Author: Yingzhen Qu
          <mailto:yqu@futurewei.com>

  Author: Jeff Tantsura
          <mailto:jefftant.ietf@gmail.com>

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

description
  "The YANG data model defines a new operation, <compare>, that
  can be used to compare NMDA datastores.";

revision 2019-07-08 {
  description
    "Initial revision";
  reference
    "RFC XXXX: Comparison of NMDA datastores";
}

/* RPC */
rpc compare {
  description
```

```
    "NMDA compare operation.";
input {
  leaf source {
    type identityref {
      base ds:datastore;
    }
    mandatory true;
    description
      "The source datastore to be compared.";
  }
  leaf target {
    type identityref {
      base ds:datastore;
    }
    mandatory true;
    description
      "The target datastore to be compared.";
  }
  leaf all {
    type empty;
    description
      "When this leaf is provided, all data nodes are compared,
      whether their schema node pertains to both datastores or
      not. When this leaf is omitted, a prefiltering step is
      automatically applied that excludes data nodes from the
      comparison that can occur in only one datastore but not
      the other. Specifically, if one of the datastores
      (source or target) contains only configuration data and
      the other datastore is <operational>, data nodes for
      which config is false are excluded from the comparison.";
  }
  choice filter-spec {
    description
      "Identifies the portions of the datastores to be
      compared.";
    anydata subtree-filter {
      description
        "This parameter identifies the portions of the
        target datastore to retrieve.";
      reference "RFC 6241, Section 6.";
    }
    leaf xpath-filter {
      if-feature nc:xpath;
      type yang:xpath1.0;
      description
        "This parameter contains an XPath expression
        identifying the portions of the target
        datastore to retrieve.";
    }
  }
}
```

```

    }
  }
}
output {
  choice compare-response {
    description
      "Comparison results.";
    leaf no-matches {
      type empty;
      description
        "This leaf indicates that the filter did not match
         anything and nothing was compared.";
    }
  }
  container differences {
    description
      "The list of differences, encoded per RFC8072 with an
       augmentation to include source values where
       applicable.";
    uses ypatch yang-patch {
      augment "yang-patch/edit" {
        description
          "Provide the value of the source of the patch,
           respectively of the comparison, in addition to
           the target value, where applicable.";
        anydata source-value {
          when "../operation = 'delete'"
            + "or ../operation = 'merge'"
            + "or ../operation = 'move'"
            + "or ../operation = 'replace'"
            + "or ../operation = 'remove'";
          description
            "The anydata 'value' is only used for 'delete',
             'move', 'merge', 'replace', and 'remove'
             operations.";
        }
      }
    }
  }
}
}
}
}
}
<CODE ENDS>
```

6. Example

The following example compares the difference between <operational> and <intended> for a subtree under "ospf". The subtree contains objects that are defined in a YANG data model for the management of OSPF defined in [I-D.ietf-ospf-yang]. The excerpt of the data model whose instantiation is basis of the comparison is as follows:

```
container ospf {  
  leaf enable {  
    type boolean;  
  }  
  leaf explicit-router-id {  
    type rt-types:router-id;  
  }  
  leaf preference {  
    type uint8;  
  }  
}
```

The contents of <intended> and <operational> datastores:

```
<ospf xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"  
  or:origin="or:intended">  
  <enable>true</enable>  
  <explicit-router-id>2.2.2.2</explicit-router-id>  
</ospf>
```

```
<ospf xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"  
  or:origin="or:operational">  
  <enable>true</enable>  
  <explicit-router-id>1.1.1.1</explicit-router-id>  
  <preference>200</preference>  
</ospf>
```

<operational> contains one object that was not contained in <intended>, "preference". Another object, "explicit-router-id", has differences in values. A third object, "enable", is the same in both cases.

RPC request to compare <operational> (source of the comparison) with <intended> (target of the comparison):

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <compare xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
    <source>ds:operational</source>
    <target>ds:intended</target>
    <xpath-filter
      xmlns:rt="urn:ietf:params:xml:ns:yang:ietf-routing"
      xmlns:ospf="urn:ietf:params:xml:ns:yang:ietf-ospf">\
        /rt:routing/rt:control-plane-protocols\
        /rt:control-plane-protocol/ospf:ospf\
      </xpath-filter>
    </compare>
  </rpc>
```

RPC reply, when a difference is detected:

```

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="101">
  <differences
    xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
    xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">
    <yang-patch>
      <patch-id>ospf router-id</patch-id>
      <comment>diff between operational and intended</comment>
      <edit>
        <edit-id>1</edit-id>
        <operation>replace</operation>
        <target>/ietf-ospf:explicit-router-id</target>
        <value>
          <ospf:explicit-router-id
            or:origin="or:system">1.1.1.1<explicit-router-id>
          </value>
          <source-value>
            <ospf:explicit-router-id
              or:origin="or:intended">2.2.2.2<explicit-router-id>
            </source-value>
          <edit-id>2</edit-id>
          <operation>create</operation>
          <target>/ietf-ospf:preference</target>
          <value>
            <ospf:preference
              or:origin="or:system">200<preference>
            </value>
          </edit>
        </yang-patch>
      </differences>
    </rpc-reply>

```

The same request in RESTCONF (using JSON format):

```

POST /restconf/operations/ietf-nmda-compare:compare HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json
Accept: application/yang-data+json

{ "ietf-nmda-compare:input" {
  "source" : "ietf-datastores:operational",
  "target" : "ietf-datastores:intended".
  "xpath-filter" : \
    "/ietf-routing:routing/control-plane-protocols\
    /control-plane-protocol/ietf-ospf:ospf"
  }
}

```

The same response in RESTCONF (using JSON format):

```

HTTP/1.1 200 OK
Date: Thu, 26 Jan 2019 20:56:30 GMT
Server: example-server
Content-Type: application/yang-data+json

{ "ietf-nmda-compare:output" : {
  "differences" : {
    "ietf-yang-patch:yang-patch" : {
      "patch-id" : "ospf router-id",
      "comment" : "diff between operational and intended",
      "edit" : [
        {
          "edit-id" : "1",
          "operation" : "replace",
          "target" : "/ietf-ospf:explicit-router-id",
          "value" : {
            "ietf-ospf:explicit-router-id" : "1.1.1.1"
            "@ietf-ospf:explicit-router-id" : {
              "ietf-origin:origin" : "ietf-origin:system"
            }
          }
          "source-value" : {
            "ietf-ospf:explicit-router-id" : "2.2.2.2"
            "@ietf-ospf:explicit-router-id" : {
              "ietf-origin:origin" : "ietf-origin:intended"
            }
          }
          "edit-id" : "2",
          "operation" : "create",
          "target" : "/ietf-ospf:preference",
          "value" : {
            "ietf-ospf:preference" : "200"
            "@ietf-ospf:preference" : {
              "ietf-origin:origin" : "ietf-origin:system"
            }
          }
        }
      ]
    }
  }
}

```

7. Open Issues

Currently, origin metadata is included in RPC output per default in comparisons that involve <operational>. It is conceivable to

introduce an input parameter that controls whether this origin metadata should in fact be included.

Currently the comparison filter is defined using subtree and XPath as in NETCONF[RFC6241]. It is not clear whether there is a requirement to allow for the definition of filters that relate instead to target resources per RESTCONF [RFC7950].

8. Possible Future Extensions

It is conceivable to extend the compare operation with a number of possible additional features in the future.

Specifically, it is possible to define an extension with an optional feature for dampening. This will allow clients to specify a minimum time period for which a difference must persist for it to be reported. This will enable clients to distinguish between differences that are only fleeting from ones that are not and that may represent a real operational issue and inconsistency within the device.

For this purpose, an additional input parameter can be added to specify the dampening period. Only differences that pertain for at least the dampening time are reported. A value of 0 or omission of the parameter indicates no dampening. Reporting of differences MAY correspondingly be delayed by the dampening period from the time the request is received.

To implement this feature, a server implementation might run a comparison when the RPC is first invoked and temporarily store the result. Subsequently, it could wait until after the end of the dampening period to check whether the same differences are still observed. The differences that still persist are then returned.

9. IANA Considerations

9.1. Updates to 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-nmda-compare

Registrant Contact: The IESG.

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

9.2. Updates to the YANG Module Names Registry

This document registers a YANG module in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registration is requested:

```
name: ietf-nmda-compare

namespace: urn:ietf:params:xml:ns:yang:ietf-nmda-compare

prefix: cp

reference: RFC XXXX
```

10. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF 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.

The RPC operation defined in this YANG module, "compare", may be considered sensitive or vulnerable in some network environments. It is thus important to control access to this operation. This is the sensitivity/vulnerability of RPC operation "compare":

Comparing datastores for differences requires a certain amount of processing resources at the server. An attacker could attempt to attack a server by making a high volume of comparison requests. Server implementations can guard against such scenarios in several ways. For one, they can implement the NETCONF access control model in order to require proper authorization for requests to be made. Second, server implementations can limit the number of requests that they serve to a client in any one time interval, rejecting requests made at a higher frequency than the implementation can reasonably sustain.

11. Acknowledgments

We thank Rob Wilton, Martin Bjorklund, Mahesh Jethanandani, Lou Berger, Kent Watsen, Phil Shafer, Ladislav Lhotka for valuable feedback and suggestions.

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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://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, <<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>>.
- [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>>.
- [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>>.
- [RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <<https://www.rfc-editor.org/info/rfc8072>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

12.2. Informative References

- [I-D.ietf-ospf-yang] Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "YANG Data Model for OSPF Protocol", draft-ietf-ospf-yang-23 (work in progress), July 2019.

Authors' Addresses

Alexander Clemm
Futurewei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ludwig@clemm.org

Yingzhen Qu
Futurewei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: yqu@futurewei.com

Jeff Tantsura
Apstra

Email: jefftant.ietf@gmail.com

Internet-Draft

July 2019

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2019

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
March 6, 2019

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-05

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 originating from an IEEE 802.1Q compliant bridge.

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.

The YANG data models in this document conforms to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

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 September 7, 2019.

Copyright Notice

Copyright (c) 2019 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
3. L3 Interface VLAN Model	4
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	24
9.1. WG version -05	24
9.2. WG version -04	24
9.3. WG version -03	24
9.4. WG version -02	24
9.5. WG version -01	24
9.6. Version -04	24
9.7. Version -03	25
10. IANA Considerations	25
11. Security Considerations	25
11.1. if-l3-vlan.yang	25
11.2. flexible-encapsulation.yang	26
12. References	28
12.1. Normative References	28
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	31
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

Tree diagrams used in this document follow the notation defined in [RFC8340].

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.

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       vlanid
        +--rw second-tag!
          |   +--rw tag-type      dot1q-tag-type
          |   +--rw vlan-id       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 an asymmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The 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      vlanid
|   |   |   |   |   |   +--rw second-tag!
|   |   |   |   |   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   |   |   |   |   +--rw vlan-id      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      vlanid
|   |   |   |   |   |   +--rw second-tag!
|   |   |   |   |   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   |   |   |   |   +--rw vlan-id      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      vlanid
|   |   |   |   |   |   +--rw second-tag!
|   |   |   |   |   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   |   |   |   |   +--rw vlan-id      vlanid
+---rw local-traffic-default-encaps!
|   +--rw outer-tag
|   |   +--rw tag-type      dot1q-tag-type

```

```
|  +--rw vlan-id      vlanid
+--rw second-tag!
  +--rw tag-type      dot1q-tag-type
  +--rw vlan-id      vlanid
```

5. L3 Interface VLAN YANG Module

This YANG module augments the encapsulation container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

```
<CODE BEGINS> file "ietf-if-l3-vlan@2019-03-05.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: Joel Jaeggli
              <mailto:joelja@gmail.com>
```

```
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 2019-03-05 {
  description "Latest draft revision";

  reference
    "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-05";
}

/*
 * 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 a 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 {
  must
    'tag-type = "dot1q-types:s-vlan" or ' +
    'tag-type = "dot1q-types:c-vlan"' {

    error-message
      "Only C-VLAN and S-VLAN tags can be matched";

    description
      "For IEEE 802.1Q interoperability, only C-VLAN and
      S-VLAN tags can be matched";
  }

  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 "Also classify on the second outermost VLAN tag";

    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 encapsulation 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@2019-03-05.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:   Joel Jaeggli
              <mailto:joelja@gmail.com>

  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 2019-03-05 {
  description "Latest draft revision";

  reference
    "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-05";
}

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;
        mandatory true;
        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 {
      must
        'tag-type = "dot1q-types:s-vlan" or ' +
        'tag-type = "dot1q-types:c-vlan"' {

        error-message
          "Only C-VLAN and S-VLAN tags can be matched";

        description
          "For IEEE 802.1Q interoperability, only C-VLAN and
           S-VLAN tags can be matched";
      }

      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 "Also classify on the second VLAN tag";

    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 {
```

```
presence
  "802.1Q tags are pushed or translated";
description "The 802.1Q tags to push (or translate if used in
            conjunction with pop-tags)";

container outer-tag {
  must
    'tag-type = "dot1q-types:s-vlan" or ' +
    'tag-type = "dot1q-types:c-vlan"' {

    error-message
      "Only C-VLAN and S-VLAN tags can be pushed";

    description
      "For IEEE 802.1Q interoperability, only C-VLAN and S-VLAN
       tags can be pushed";
  }

  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
    "In addition to the first tag, also push/rewrite a second
     VLAN tag.";

  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 {
        must
            'tag-type = "dot1q-types:s-vlan" or ' +
            'tag-type = "dot1q-types:c-vlan"' {

            error-message
                "Only C-VLAN and S-VLAN tags can be matched";

            description
                "For IEEE 802.1Q interoperability, only C-VLAN and
                S-VLAN tags can be matched";
        }

        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::1</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 classified 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

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

9.1. WG version -05

- o Incorporate feedback from IEEE 802.1 WG, John Messenger in particular.
- o Adding must constraints to ensure outer tags are always matched to C-VLAN and S-VLAN tags.
- o Fixed bug where second tag could be matched without outer tag, and where tags must not be specified.

9.2. WG version -04

- o Added examples

9.3. WG version -03

- o Fix namespace bug in XPath identity references, removed extraneous 'dot1q-tag' containers.

9.4. WG version -02

- o Use explicit containers for outer and inner tags rather than lists.

9.5. WG version -01

- o Tweaked the abstract.
- o Removed unnecessary feature for the L3 sub-interface module.
- o Update the 802.1Qcp type references.
- o Remove extra tag container for L3 sub-interfaces YANG.

9.6. Version -04

- o IEEE 802.1 specific types have been removed from the draft. These are now referenced from the 802.1Qcp draft YANG modules.
- o Fixed errors in the xpath expressions.

9.7. Version -03

- o Incorporates feedback received from presenting to the IEEE 802.1 WG.
- o Updates the modules for double tag matches/rewrites to restrict the outer tag type to S-VLAN and inner tag type to C-VLAN.
- o Updates the introduction to indicate primary use case is for IETF forwarding protocols.
- o Updates the objectives to make IEEE 802.1Q bridge interoperability a key objective.

10. IANA Considerations

This document defines two 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:

- o outer-tag/tag-type
- o outer-tag/vlan-id
- o second-tag/tag-type
- o 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:

- o default
- o untagged
- o dot1q-priority-tagged
- o dot1q-priority-tagged/tag-type
- o dot1q-vlan-tagged/outer-tag/vlan-type
- o dot1q-vlan-tagged/outer-tag/vlan-id
- o dot1q-vlan-tagged/second-tag/vlan-type
- o dot1q-vlan-tagged/second-tag/vlan-id

There are also many nodes in the flexible-encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/rewrite, that are sensitive to this are:

- o symmetrical/dot1q-tag-rewrite/pop-tags
- o symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o symmetrical/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- o symmetrical/dot1q-tag-rewrite/push-tags/second-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- o 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:

- o outer-tag/vlan-type
- o outer-tag/vlan-id
- o second-tag/vlan-type
- o 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-07 (work in progress), March 2019.
- [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., "IEEE 802.1Qcp-2018 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-09 (work in progress), October 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>>.
- [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>>.

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 has developed a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- o The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.
- o Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.
- o Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.
- o In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.
- o The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- o Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.
- o Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.
- o Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.

- o VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.
- o Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q – other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Netmod
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2020

B. Lengyel
Ericsson
B. Claise
Cisco Systems, Inc.
July 5, 2019

YANG Instance Data File Format
draft-ietf-netmod-yang-instance-file-format-03

Abstract

There is a need to document data defined in YANG models when a live server is not available. Data is often needed already at design or implementation time or needed by groups that do not have a live running server available. This document specifies a standard file format for YANG instance data (which follows the syntax and semantic from existing YANG models, re-using the same format as the reply to a <get> operation/request) and decorates it with metadata.

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 6, 2020.

Copyright Notice

Copyright (c) 2019 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. Terminology	2
2. Introduction	3
2.1. Principles	4
3. Instance Data File Format	4
3.1. Specifying the Content Schema	6
3.1.1. INLINE Method	7
3.1.2. URI Method	7
3.2. Examples	8
4. Data Life cycle	11
5. Delivery of Instance Data	12
6. Backwards Compatibility	12
7. Yang Instance Data Model	12
7.1. Tree Diagram	12
7.2. YANG Model	13
8. Security Considerations	17
9. IANA Considerations	18
9.1. URI Registration	18
9.2. YANG Module Name Registration	18
10. Acknowledgments	18
11. References	18
11.1. Normative References	18
11.2. Informative References	20
Appendix A. Open Issues	20
Appendix B. Changes between revisions	20
Appendix C. Detailed Use Cases – Non-Normative	23
C.1. Use Cases	23
C.1.1. Use Case 1: Early Documentation of Server Capabilities	23
C.1.2. Use Case 2: Preloading Data	24
C.1.3. Use Case 3: Documenting Factory Default Settings	24
Authors' Addresses	24

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.

Instance Data Set: A named set of data items decorated with metadata that can be used as instance data in a YANG data tree.

Instance Data File: A file containing an instance data set formatted according to the rules described in this document.

Content-schema: A set of YANG modules with their revision, supported features and deviations for which the instance data set contains instance data

Content defining Yang module(s): YANG module(s) that make up the content-schema

YANG Instance Data, or just instance data for short, is data that could be stored in a datastore and whose syntax and semantics is defined by YANG models.

The term Server is used as defined in [RFC8342]

2. Introduction

There is a need to document data defined in YANG models when a live server is not available. Data is often needed already at design or implementation time or needed by groups that do not have a live running server available. To facilitate this off-line delivery of data this document specifies a standard format for YANG instance data sets and YANG instance data files.

The following is a list of already implemented and potential use cases.

- UC1 Documentation of server capabilities
- UC2 Preloading default configuration data
- UC3 Documenting Factory Default Settings
- UC4 Instance data used as backup
- UC5 Storing the configuration of a device, e.g. for archive or audit purposes
- UC6 Storing diagnostics data
- UC7 Allowing YANG instance data to potentially be carried within other IPC message formats
- UC8 Default instance data used as part of a templating solution
- UC9 Providing data examples in RFCs or internet drafts

In Appendix C we describe the first three use cases in detail.

There are many and varied use cases where YANG instance data could be used. We do not want to limit future uses of instance data sets, so specifying how and when to use Yang instance data is out of scope for this document. It is anticipated that other documents will define specific use cases. Use cases are listed here only to indicate the need for this work.

2.1. Principles

The following is a list of the basic principles of the instance data format:

- P1 Two standard formats are based on the XML and the JSON encoding
- P2 Re-use existing formats similar to the response to a <get> operation/request
- P3 Add metadata about the instance data set (Section 3, Paragraph 9)
- P4 A YANG instance data set may contain data for many YANG modules
- P5 Instance data may include configuration data, state data or a mix of the two
- P6 Partial data sets are allowed
- P7 YANG instance data format may be used for any data for which YANG module(s) are defined and available to the reader, independent of whether the module is actually implemented by a server

3. Instance Data File Format

A YANG instance data file MUST contain a single instance data set and no additional data.

The format of the instance data set is defined by the ietf-yang-instance-data YANG module. It is made up of a header part and content-data. The header part carries metadata for the instance data set. The content-data, defined as an anydata data node, carries the "real data" that we want to document/provide. The syntax and semantics of content-data is defined by the content-schema.

Two formats are specified based on the XML and JSON YANG encodings. Later as other YANG encodings (e.g. CBOR) are defined further instance data formats may be specified.

The content-data part SHALL follow the encoding rules defined in [RFC7950] for XML and [RFC7951] for JSON and MUST use UTF-8 character encoding. Content-data MAY include:

metadata as defined by [RFC7952].

entity-tags and timestamps as defined in [RFC8040] encoded according to [RFC7952]

a default attribute as defined in [RFC6243] section 6. and in [RFC8040] section 4.8.9.

origin metadata as specified in [RFC8526] and [RFC8527]

implementation specific metadata. Unknown metadata MUST be ignored by users of YANG instance data, allowing it to be used later for other purposes.

in the XML format implementation specific XML attributes. Unknown attributes MUST be ignored by users of YANG instance data, allowing them to be used later for other purposes.

The content-data part will be very similar to the result returned for a NETCONF <get-data> or for a RESTCONF get operation.

The content-data part MUST conform to the content-schema. An instance data set MAY contain data for any number of YANG modules; if needed it MAY carry the complete configuration and state data set for a server. Default values SHOULD NOT be included.

Config=true and config=false data MAY be mixed in the instance data file.

Instance data files MAY contain partial data sets. This means mandatory, min-elements, require-instance=true, must and when constrains MAY be violated.

The name of the instance data file SHOULD take one of the following two forms:

If revision information inside the data set is present

* instance-data-set-name ['@' revision-date] '.filetype'

* E.g. acme-router-modules@2018-01-25.xml

If the leaf name is present in the instance data header this MUST be used. Revision-date MUST be set to the latest revision date inside the instance data set.

If timestamp information inside the data set is present

* instance-data-set-name ['@' timestamp] '.filetype'

* E.g. acme-router-modules@2018-01-25T15_06_34_3+01_00.json

If the leaf name is present in the instance data header this MUST be used. If the leaf timestamp is present in the instance data header this MUST be used; the semicolons and the decimal point if present shall be replaced by underscores.

The revision date or timestamp is optional. ".filetype" SHALL be ".json" or ".xml" according to the format used.

Metadata, information about the data set itself SHOULD be included in the instance data set. Some metadata items are defined in the YANG module ietf-yang-instance-data, but other items MAY also be used. Metadata SHOULD include:

- o Name of the data set
- o Content schema specification
- o Description of the instance data set. The description SHOULD contain information whether and how the data can change during the lifetime of the server.

3.1. Specifying the Content Schema

To properly understand and use an instance data set the user needs to know the content-schema. One of the following methods SHOULD be used:

INLINE method: Include the needed information as part of instance data set.

URI method: Include a URI that references another YANG instance data file. This instance data file will use the same content-schema as the referenced YANG instance data file. (if you don't want to repeat the info again and again)

EXTERNAL Method: Do not include the content-schema as it is already known, or the information is available through external documents.

Additional methods e.g. a YANG-package based solution may be added later.

Note, the specified content-schema only indicates the set of modules that were used to define this YANG instance data set. Sometimes instance data may be used for a server supporting a different YANG module set. (e.g. for "UC2 Preloading Data" the instance data set may not be updated every time the YANG modules on the server are updated) Whether the instance data set is usable for a possibly different real-life YANG module set depends on many factors including the compatibility between the specified and the real-life YANG module set (considering modules, revisions, features, deviations), the scope of the instance data, etc.

3.1.1.1. INLINE Method

One or more inline-target-spec elements define YANG module(s) used to specify the content defining YANG modules.

E.g. ietf-yang-library@2016-06-21.yang

The anydata inline-content-schema carries instance data (conforming to the inline-target-spec modules) that actually specifies the content defining YANG modules including revision, supported features, deviations and any relevant additional data (e.g. version labels)

3.1.1.2. URI Method

A schema-uri leaf SHALL contain a URI that references another YANG instance data file. The current instance data file will use the same content schema as the referenced file.

The referenced instance data file MAY have no content-data if it is used solely for specifying the content-schema. The referenced YANG instance data file might use the INLINE method or might use the URI method to reference further instance data file(s). However at the end of this reference chain there MUST be an instance data file using the INLINE method.

If a referenced instance data file is not available the revision data, supported features and deviations for the target YANG modules are unknown.

The URI method is advantageous when the user wants to avoid the overhead of specifying the content-schema in each instance data file: E.g. In Use Case 6, when the system creates a diagnostic file every minute to document the state of the server.

3.2. Examples

The following example is based on "UC1, Documenting Server Capabilities". It provides (a shortened) list of supported YANG modules and Netconf capabilities for a server. It uses the inline method to specify the content-schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<instance-data-set xmlns=
  "urn:ietf:params:xml:ns:yang:ietf-yang-instance-data">
  <name>acme-router-modules</name>
  <inline-spec>
    ietf-yang-library@2016-06-21.yang
  </inline-spec>
  <inline-content-schema>
    <module-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
      <module>
        <name>ietf-yang-library</name>
        <revision>2016-06-21</revision>
      </module>
      <module>
        <name>ietf-netconf-monitoring</name>
        <revision>2010-10-04</revision>
      </module>
    </module-state>
  </inline-content-schema>
  <revision>
    <date>1956-10-23</date>
    <description>Initial version</description>
  </revision>
  <description>Defines the minimal set of modules that any acme-router
    will contain.</description>
  <contact>info@acme.com</contact>
  <content-data>
    <!-- The example lists only 4 modules, but it could list the
      full set of supported modules for a server, potentially many
      dozens of modules -->
    <module-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
      <module>
        <name>ietf-yang-library</name>
        <revision>2016-06-21</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-yang-library
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-system</name>
```

```
<revision>2014-08-06</revision>
<namespace>urn:ietf:params:xml:ns:yang:ietf-system</namespace>
<feature>sys:authentication</feature>
<feature>sys:local-users</feature>
<deviation>
  <name>acme-system-ext</name>
  <revision>2018-08-06</revision>
</deviation>
<conformance-type>implement</conformance-type>
</module>
<module>
  <name>ietf-yang-types</name>
  <revision>2013-07-15</revision>
  <namespace>urn:ietf:params:xml:ns:yang:ietf-yang-types
    </namespace>
  <conformance-type>import</conformance-type>
</module>
<module>
  <name>acme-system-ext</name>
  <revision>2018-08-06</revision>
  <namespace>urn:rdns:acme.com:oammodel:acme-system-ext
    </namespace>
  <conformance-type>implement</conformance-type>
</module>
</module-state>
<netconf-state>
  <capabilities>
    <capability>
      urn:ietf:params:netconf:capability:validate:1.1
    </capability>
  </capabilities>
</netconf-state>
</content-data>
</instance-data-set>
```

Figure 1: XML Instance Data Set - Use case 1, Documenting server capabilities

The following example is based on "UC2, Preloading Default Configuration". It provides a (shortened) default rule set for a read-only operator role. It uses the inline method for specifying the content-schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<instance-data-set xmlns=
  "urn:ietf:params:xml:ns:yang:ietf-yang-instance-data">
  <name>read-only-acm-rules</name>
  <inline-spec>ietf-yang-library@2019-01-04.yang</inline--spec>
  <inline-content-schema>
    <yang-library xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
      <module-set>
        <name>all</name>
        <module>
          <name>ietf-netconf-acm</name>
          <revision>2012-02-22</revision>
        </module>
      </module-set>
    </yang-library>
  </inline-content-schema>
  <revision>
    <date>1776-07-04</date>
    <description>Initial version</description>
  </revision>
  <description>Access control rules for a read-only role.</description>
  <content-data>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
      <enable-nacm>true</enable-nacm>
      <read-default>deny</read-default>
      <exec-default>deny</exec-default>
      <rule-list>
        <name>read-only-role</name>
        <group>read-only-group</group>
        <rule>
          <name>read-all</name>
          <module-name>*</module-name>
          <access-operation>read</access-operation>
          <action>permit</action>
        </rule>
      </rule-list>
    </nacm>
  </content-data>
</instance-data-set>

```

Figure 2: XML Instance Data Set - Use case 2, Preloading access control data

The following example is based on UC6 Storing diagnostics data. An instance data set is produced by the server every 15 minutes that contains statistics about NETCONF. As a new set is produced periodically many times a day a revision-date would be useless; instead a timestamp is included.

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "acme-router-netconf-diagnostics",
    "schema-uri": "file:///acme-netconf-diagnostics-yanglib.json",
    "timestamp": "2018-01-25T17:00:38Z",
    "description":
      "Netconf statistics",
    "content-data": {
      "ietf-netconf-monitoring:netconf-state": {
        "statistics": {
          "netconf-start-time ": "2018-12-05T17:45:00Z",
          "in-bad-hellos ": "32",
          "in-sessions ": "397",
          "dropped-sessions ": "87",
          "in-rpcs ": "8711",
          "in-bad-rpcs ": "408",
          "out-rpc-errors ": "408",
          "out-notifications": "39007"
        }
      }
    }
  }
}
```

Figure 3: JSON Instance Data File example – UC6 Storing diagnostics data

4. Data Life cycle

In UC2 "Preloading default configuration data" the loaded data may be changed later e.g. by management operations. In UC6 "Storing Diagnostics data" the diagnostics values may change on device every second.

YANG instance data is a snap-shot of information at a specific point of time. If the data changes afterwards this is not represented in the instance data set anymore. The valid values can be retrieved in run-time via NETCONF/RESTCONF or received e.g. in Yang-Push notifications.

Whether the instance data changes and if so, when and how, SHOULD be described either in the instance data set's description statement or in some other implementation specific manner.

5. Delivery of Instance Data

Instance data sets that are produced as a result of some sort of specification or design effort SHOULD be available without the need for a live server e.g. via download from the vendor's website, or in any other way product documentation is distributed.

Other instance data sets may be read from or produced by the YANG server itself e.g. UC6 documenting diagnostic data.

6. Backwards Compatibility

The concept of backwards compatibility and what changes are backwards compatible are not defined for instance data sets as it is highly dependent on the specific use case and the content-schema.

For instance data that is the result of a design or specification activity some changes that may be good to avoid are listed. YANG uses the concept of managed entities identified by key values; if the connection between the represented entity and the key value is not preserved during an update this may lead to problems.

- o If the key value of a list entry that represents the same managed entity as before is changed, the user may mistakenly identify the list entry as new.
- o If the meaning of a list entry is changed, but the key values are not (e.g. redefining an alarm-type but not changing its alarm-type-id) the change may not be noticed.
- o If the key value of a previously removed list entry is reused for a different entity, the change may be mis-interpreted as reintroducing the previous entity.

7. Yang Instance Data Model

7.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model.


```

module: ietf-yang-instance-data
  structure instance-data-set:
    +--rw name? string
    +--rw (content-schema-spec)?
      | +--:(inline)
      | | +--rw inline-spec* string
      | | +--rw inline-content-schema <anydata>
      | +--:(uri)
      | | +--rw schema-uri? inet:uri
    +--rw description? string
    +--rw contact? string
    +--rw organization? string
    +--rw datastore? ds:datastore-ref
    +--rw revision* [date]
      | +--rw date string
      | +--rw description? string
    +--rw timestamp? yang:date-and-time
    +--rw content-data? <anydata>

```

7.2. YANG Model

```

<CODE BEGINS> file "ietf-yang-instance-data@2019-07-04.yang"
module ietf-yang-instance-data {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-instance-data";
  prefix yid ;

  import ietf-yang-structure-ext { prefix sx; }
  import ietf-datastores { prefix ds; }
  import ietf-inet-types { prefix inet; }
  import ietf-yang-types { prefix yang; }
  import ietf-yang-metadata { prefix "md"; }

  organization "IETF NETMOD Working Group";
  contact
    "WG Web: <https://datatracker.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>

    Author: Balazs Lengyel
      <mailto:balazs.lengyel@ericsson.com>";

  description "The module defines the structure and content of YANG
    instance data sets.

    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) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2019 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

```
revision 2019-07-04 {
  description "Initial revision.";
  reference "RFC XXXX: YANG Instance Data Format";
}

md:annotation entity-tag {
  type string;
  description "Used to encode the entity-tag defined in
    RFC8040 for the annotated instance.";
  reference "RESTCONF Protocol RFC8040";
}

md:annotation last-modified {
  type yang:date-and-time;
  description "Contains the date and time when the annotated
    instance was last modified (or created).";
  reference "RESTCONF Protocol RFC8040";
}

sx:structure instance-data-set {
  description "A data structure to define a format for a
    YANG instance data set. Consists of meta-data about
    the instance data set and the real content-data.";

  leaf name {
    type string;
    description "Name of the YANG instance data set.";
  }

  choice content-schema-spec {
```

```
description "Specification of the content-schema";

case inline {
  leaf-list inline-spec {
    type string {
      pattern '.*@{d{4}}-{d{2}}-{d{2}}\.yang';
    }
    min-elements 1;
    ordered-by user;
    description
      "Indicates that content defining Yang modules
       are specified inline.
       Each value MUST be a YANG Module name including the
       revision-date as defined for YANG file names in RFC7950.

       E.g. ietf-yang-library@2016-06-21.yang

       The first item is either ietf-yang-library or some other
       YANG module that contains a list of YANG modules with
       their name, revision-date, supported-features and
       deviations.
       As some versions of ietf-yang-library MAY contain
       different module-sets for different datastores, if
       multiple module-sets are included, the instance data set's
       meta-data MUST contain the datastore information and
       instance data for the ietf-yang-library MUST also contain
       information specifying the module-set for the relevant
       datastore.

       Subsequent items MAY specify YANG modules augmenting the
       first module with useful data (e.g. a version label).";
  }
  anydata inline-content-schema {
    mandatory true;
    description "Instance data corresponding to the YANG modules
      specified in the inline-spec nodes defining the set
      of content defining Yang YANG modules for this
      instance-data-set.";
  }
}

case uri {
  leaf schema-uri {
    type inet:uri;
    description
      "A reference to another YANG instance data file.
      This instance data file will use the same set of target
      YANG modules, revisions, supported features and deviations
```

```
        as the referenced YANG instance data file.";
    }
}

leaf-list description {
    type string;
    description "Description of the instance data set.";
}

leaf contact {
    type string;
    description "Contact information for the person or
        organization to whom queries concerning this
        instance data set should be sent.";
}

leaf organization {
    type string;
    description "Organization responsible for the instance
        data set.";
}

leaf datastore {
    type ds:datastore-ref;
    description "The identity of the datastore with which the
        instance data set is associated e.g. the datastore from
        where the data was read or the datastore where the data
        could be loaded or the datastore which is being documented.
        If a single specific datastore can not be specified, the
        leaf MUST be absent.

        If this leaf is absent, then the datastore to which the
        instance data belongs is undefined.";
}

list revision {
    key date;
    description "Instance data sets that are produced as
        a result of some sort of specification or design effort
        SHOULD have at least one revision entry. For every
        published editorial change, a new one SHOULD be added
        in front of the revisions sequence so that all
        revisions are in reverse chronological order.

        For instance data sets that are read from
        or produced by a server or otherwise
        subject to frequent updates or changes, revision
```

```
        SHOULD NOT be present";

    leaf date {
        type string {
            pattern '\d{4}-\d{2}-\d{2}';
        }
        description "Specifies the date the instance data set
            was last modified. Formatted as YYYY-MM-DD";
    }

    leaf description {
        type string;
        description
            "Description of this revision of the instance data set.";
    }
}

leaf timestamp {
    type yang:date-and-time;
    description "The date and time when the instance data set
        was last modified.

        For instance data sets that are read from or produced
        by a server or otherwise subject to frequent
        updates or changes, timestamp SHOULD be present";
}

anydata content-data {
    description "Contains the real instance data.
        The data MUST conform to the relevant YANG Modules specified
        either in the content-schema-spec or in some other
        implementation specific manner.";
}
}
}
<CODE ENDS>
```

8. Security Considerations

Depending on the nature of the instance data, instance data files MAY need to be handled in a secure way. The same type of handling should be applied, that would be needed for the result of a <get> operation returning the same data.

9. IANA Considerations

This document registers one URI and one YANG module.

9.1. URI Registration

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-instance-data

Registrant Contact: The IESG.

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

9.2. YANG Module Name Registration

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

name: ietf-yang-instance-data
namespace: urn:ietf:params:xml:ns:yang:ietf-yang-instance-data
prefix: yid
reference: RFC XXXX

10. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Juergen Schoenwaelder, Rob Wilton, Joe Clarke, Kent Watsen, Martin Bjorklund, Ladislav Lhotka, Qin Wu and other members of the Netmod WG.

11. References

11.1. Normative References

- [I-D.ietf-netmod-yang-data-ext]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", draft-ietf-netmod-yang-data-ext-03 (work in progress), April 2019.
- [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>>.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <<https://www.rfc-editor.org/info/rfc6243>>.
- [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>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <<https://www.rfc-editor.org/info/rfc8526>>.
- [RFC8527] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", RFC 8527, DOI 10.17487/RFC8527, March 2019, <<https://www.rfc-editor.org/info/rfc8527>>.

11.2. Informative References

- [I-D.ietf-ccamp-alarm-module]
Vallin, S. and M. Bjorklund, "YANG Alarm Module", draft-ietf-ccamp-alarm-module-09 (work in progress), April 2019.
- [I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-07 (work in progress), October 2018.
- [I-D.ietf-netconf-yang-push]
Clemm, A. and E. Voit, "Subscription to YANG Datastores", draft-ietf-netconf-yang-push-25 (work in progress), May 2019.
- [I-D.wu-netconf-restconf-factory-restore]
Wu, Q., Lengyel, B., and Y. Niu, "Factory default Setting", draft-wu-netconf-restconf-factory-restore-03 (work in progress), October 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>>.
- [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>>.

Appendix A. Open Issues

- o -

Appendix B. Changes between revisions

v02 - v03

- o target renamed to "content-schema" and "content defining Yang module(s)"
- o Made name of instance data set optional
- o Updated according to draft-ietf-netmod-tang-data-ext-02
- o Clarified that entity-tag and last-modified timestamp are encoded as metadata. While they contain useful data, the HTTP-header based encoding from Restconf is not suitable.

- o

v01 - v02

- o Removed design time from terminology
- o Defined the format of the content-data part by referencing various RFCs and drafts instead of the result of the get-data and get operations.
- o Changed target-ptr to a choice
- o Inline target-ptr may include augmenting modules and alternatives to ietf-yang-library
- o Moved list of target modules into a separate <target-modules> element.
- o Added backwards compatibility considerations

v00 - v01

- o Added the target-ptr metadata with 3 methods
- o Added timestamp metadata
- o Removed usage of dedicated .yid file extension
- o Added list of use cases
- o Added list of principles
- o Updated examples
- o Moved detailed use case descriptions to appendix

v05 - v00-netmod

- o New name for the draft following Netmod workgroup adoption. No other changes

v04 - v05

- o Changed title and introduction to clarify that this draft is only about the file format and documenting server capabilities is just a use case.
- o Added reference to draft-wu-netconf-restconf-factory-restore

- o Added new open issues.

v03 - v04

- o Updated changelog for v02-v03

v02 - v03

- o Updated the document according to comments received at IETF102
- o Added parameter to specify datastore
- o Rearranged chapters
- o Added new use case: Documenting Factory Default Settings
- o Added "Target YANG Module" to terminology
- o Clarified that instance data is a snapshot valid at the time of creation, so it does not contain any later changes.
- o Removed topics from Open Issues according to comments received at IETF102

v01 - v02

- o The recommendation to document server capabilities was changed to be just the primary use-case. (Merged chapter 4 into the use case chapter.)
- o Stated that RFC7950/7951 encoding must be followed which also defines (dis)allowed whitespace rules.
- o Added UTF-8 encoding as it is not specified in t950 for instance data
- o added XML declaration

v00 - v01

- o Redefined using yang-data-ext
- o Moved metadata into ordinary leafs/leaf-lists

Appendix C. Detailed Use Cases - Non-Normative

C.1. Use Cases

We present a number of use cases where YANG instance data is needed.

C.1.1. Use Case 1: Early Documentation of Server Capabilities

A server has a number of server-capabilities that are defined in YANG modules and can be retrieved from the server using protocols like NETCONF or RESTCONF. server capabilities include

- o data defined in ietf-yang-library: YANG modules, submodules, features, deviations, schema-mounts, datastores supported ([I-D.ietf-netconf-rfc7895bis])
- o alarms supported ([I-D.ietf-ccamp-alarm-module])
- o data nodes, subtrees that support or do not support on-change notifications ([I-D.ietf-netconf-yang-push])
- o netconf-capabilities in ietf-netconf-monitoring

While it is good practice to allow a client to query these capabilities from the live server, that is often not possible.

Often when a network node is released an associated NMS (network management system) is also released with it. The NMS depends on the capabilities of the server. During NMS implementation information about server capabilities is needed. If the information is not available early in some off-line document, but only as instance data from the live network node, the NMS implementation will be delayed, because it has to wait for the network node to be ready. Also assuming that all NMS implementors will have a correctly configured network node available to retrieve data from, is a very expensive proposition. (An NMS may handle dozens of node types.)

Network operators often build their own home-grown NMS systems that need to be integrated with a vendor's network node. The operator needs to know the network node's server capabilities in order to do this. Moreover the network operator's decision to buy a vendor's product may even be influenced by the network node's OAM feature set documented as the Server's capabilities.

Beside NMS implementors, system integrators and many others also need the same information early. Examples could be model driven testing, generating documentation, etc.

Most server-capabilities are relatively stable and change only during upgrade or due to licensing or addition or removal of HW. They are usually defined by a vendor at design time, before the product is released. It is feasible and advantageous to define/document them early e.g. in a YANG instance data File.

It is anticipated that a separate IETF document will define in detail how and which set of server capabilities should be documented.

C.1.2. Use Case 2: Preloading Data

There are parts of the configuration that must be fully configurable by the operator, however for which often a simple default configuration will be sufficient.

One example is access control groups/roles and related rules. While a sophisticated operator may define dozens of different groups often a basic (read-only operator, read-write system administrator, security-administrator) triplet will be enough. Vendors will often provide such default configuration data to make device configuration easier for an operator.

Defining Access control data is a complex task. To help the device vendor pre-define a set of default groups (/nacm:nacm/groups) and rules for these groups to access specific parts of common models (/nacm:nacm/rule-list/rule).

YANG instance data files are used to document and/or preload the default configuration.

C.1.3. Use Case 3: Documenting Factory Default Settings

Nearly every server has a factory default configuration. If the system is really badly misconfigured or if the current configuration is to be abandoned the system can be reset to this default.

In Netconf the <delete-config> operation can already be used to reset the startup datastore. There are ongoing efforts to introduce a new, more generic reset-datastore operation for the same purpose [I-D.wu-netconf-restconf-factory-restore]

The operator currently has no way to know what the default configuration actually contains. YANG instance data can be used to document the factory default configuration.

Authors' Addresses

Balazs Lengyel
Ericsson
Magyar Tudosok korutja 11
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2020

J. Clarke, Ed.
Cisco Systems, Inc.
July 3, 2019

YANG Module Versioning Requirements
draft-ietf-netmod-yang-versioning-reqs-01

Abstract

This document describes the problems that can arise because of the YANG language module update rules, that require all updates to YANG module preserve strict backwards compatibility. It also defines the requirements on any solution designed to solve the stated problems. This document does not consider possible solutions, nor endorse any particular solution.

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 4, 2020.

Copyright Notice

Copyright (c) 2019 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. Background	2
2.1. Striving for model perfection	3
2.2. Some YANG Modules Are Not Backwards-Compatible	3
2.3. Non-Backwards-Compatible Errors	4
2.4. No way to easily decide whether a change is Backwards-Compatible	4
2.5. No good way to specify which module revision to import	5
2.6. Early Warning about Removal	6
2.7. Clear Indication of Node Support	6
3. Terminology and Conventions	7
4. The Problem Statement	7
5. Requirements of a YANG Versioning Solution	9
6. Contributors	11
7. Acknowledgments	11
8. Security Considerations	11
9. IANA Considerations	12
10. References	12
10.1. Normative References	12
10.2. Informative References	12
Author's Address	12

1. Introduction

This requirements document initially considers some of the existing YANG module update rules, then describes the problems that arise due to those rules embracing strict backwards compatibility, and finally defines requirements on any solution that may be designed to solve these problems by providing an alternative YANG versioning strategy.

2. Background

The YANG data modeling language [RFC7950] specifies strict rules for updating YANG modules (see section 11 "Updating a Module"). Citing a few of the relevant rules:

1. "As experience is gained with a module, it may be desirable to revise that module. However, changes to published modules are not allowed if they have any potential to cause interoperability problems between a client using an original specification and a server using an updated specification."

2. "Note that definitions contained in a module are available to be imported by any other module and are referenced in "import" statements via the module name. Thus, a module name MUST NOT be changed. Furthermore, the "namespace" statement MUST NOT be changed, since all XML elements are qualified by the namespace."
3. "Otherwise, if the semantics of any previous definition are changed (i.e., if a non-editorial change is made to any definition other than those specifically allowed above), then this MUST be achieved by a new definition with a new identifier."
4. "deprecated indicates an obsolete definition, but it permits new/continued implementation in order to foster interoperability with older/existing implementations."

The rules described above, along with other similar rules, causes various problems, as described in the following sections:

2.1. Striving for model perfection

The points made above lead to the logical conclusion that the standardized YANG modules have to be perfect on day one (at least the structure and meaning), which in turn might explain why IETF YANG modules take so long to standardize. Shooting for perfection is obviously a noble goal, but if the perfect standard comes too late, it doesn't help the industry.

2.2. Some YANG Modules Are Not Backwards-Compatible

As we learn from our mistakes, we're going to face more and more non-backwards-compatible YANG modules. An example is the YANG data model for L3VPN service delivery [RFC8049], which, based on implementation experience, has been updated in a non-backwards-compatible way by [RFC8299].

While Standards Development Organization (SDO) YANG modules are obviously better for the industry, we must recognize that many YANG modules are actually generated YANG modules (for example, from internal databases), which is sometimes the case for vendor modules [RFC8199]. From time to time, the new YANG modules are not backwards-compatible.

Old module parts that are no longer needed, no longer supported, or are not used by consumers need to be removed from modules. It is often hard to decide which parts are no longer needed/used; still the need and practice of removing old parts exist. While it is rare in standard modules it is more common in vendor YANG modules where the usage of modules is more controlled.

The problems described in Section 2.7 may also result in incompatible changes.

In such cases, it would be better to indicate how backwards-compatible a given YANG module actually is.

As modules are sometimes updated in an incompatible way the current assumption that once a YANG module is defined all further revisions can be freely used as they are compatible is not valid.

2.3. Non-Backwards-Compatible Errors

Sometimes small errors force us to make non-backwards-compatible updates. As an example imagine that we have a string with a complex pattern (e.g., an IP address). Let's assume the initial pattern incorrectly allows IP addresses to start with 355. In the next version this is corrected to disallow addresses starting with 355. Formally this is a non-backwards-compatible change as the value space of the string is decreased. In reality an IP address and the implementation behind it was never capable of handling an address starting with 355. So practically this is a backwards-compatible change, just like a correction of the description statement. Current YANG rules are ambiguous as to whether non-backwards-compatible bug fixes are allowed without also requiring a module name change.

2.4. No way to easily decide whether a change is Backwards-Compatible

A management system, SDN controller, or any other user of a module should be capable of easily determining the compatibility between two module versions. Higher level logic for a network function, something that cannot be implemented in a purely model driven way, is always dependent on a specific version of the module. If the client finds that the module has been updated on the network node, it has to decide if it tries to handle it as it handled the previous version of the model or if it just stops, to avoid problems. To make this decision the client needs to know if the module was updated in a backwards-compatible way or not.

This is not possible to decide today because of the following:

- o It is sometimes necessary to change the semantic behavior of a data node, action or rpc while the YANG definition does not change (with the possible exception of the description statement). In such a case it is impossible to determine whether the change is backwards-compatible just by looking at the YANG statements. It's only the human model designer who can decide.

- o Problems with the deprecated and obsolete status statement, Section 2.7
- o YANG module authors might decide to violate YANG 1.1 update rules for some of the reasons above.

Finding status changes or violations of update rules need a line-by-line comparison of the old and new modules is a tedious task.

2.5. No good way to specify which module revision to import

If a module (MOD-A) is imported by another one (MOD-B) the importer may specify which revision must be imported. Even if MOD-A is updated in a backwards-compatible way not all revisions will be suitable, e.g., a new MOD-B might need the newest MOD-A. However, both specifying or omitting the revision date for import leads to problems.

If the import by revision-date is specified

- o If corrections are made to MOD-A these would not have any effect as the import's revision date would still point to the uncorrected earlier YANG module revision.
- o If MOD-A is updated in a backwards-compatible way because another importer (MOD-C) needs some functionality, the new MOD-A could be used by MOD-B, but specifying the exact import revision-date prevents this. This will force the implementers to import two different revisions of MOD-A, forcing them to maintain old MOD-A revisions unnecessarily.
- o If multiple modules import different revisions of MOD-A the human user will need to understand the subtle differences between the different revisions. Small differences would easily lead to operator mistakes as the operator will rarely check the documentation.
- o Tooling/SW is often not prepared to handle multiple revisions of the same YANG module.

If the import revision-date is not specified

- o any revision of MOD-A may be used including unsuitable ones. Older revisions may be lacking functionality MOD-B needs. Newer MOD-A revisions may obsolete definitions used by MOD-B in which case these must not be used by MOD-B anymore.

- o As it is not specified which revisions of MOD-A are suitable for MOD-B. The problem has to be solved on a case by case basis studying all the details of MOD-A and MOD-B which is considerable work.

2.6. Early Warning about Removal

If a schema part is considered old/bad we need to be able to give advance warning that it will be removed. As this is an advance warning the part must still be present and usable in the current revision; however, it will be removed in one of the next revisions. The deprecated statement cannot be reliably used for this purpose both because deprecated nodes may not be implemented and also there is no mandate that text be provided explaining the deprecation.

We need the advance warning to allow users of the module time to plan/execute migration away from the deprecated functionality. Deprecation should be accompanied by information whether the functionality will just disappear or that there is an alternative, possibly more advanced solution that should be used.

Vendors use such warnings often, but the NMDA related redesign of IETF modules is also an example where it would be useful for IETF. As another example, see the usage of deprecated in the Java programming language.

2.7. Clear Indication of Node Support

The current definition of deprecated and obsolete in [RFC7950] (as quoted below) is problematic and should be corrected.

- o "deprecated" indicates an obsolete definition, but it permits new/continued implementation in order to foster interoperability with older/existing implementations.
- o "obsolete" means that the definition is obsolete and SHOULD NOT be implemented and/or can be removed from implementations.

YANG is considered an interface contract between the server and the client. The current definitions of deprecated and obsolete mean that a schema node that is either deprecated or obsolete may or may not be implemented. The client has no way to find out which is the case except for by trying to write or read data at the leaf in question. This probing would need to be done for each separate data-node, which is not a trivial thing to do. This "may or may not" is unacceptable in a contract. In effect, this works as if there would be an if-feature statement on each deprecated schema node where the server

does not advertise whether the feature is supported or not. Why is it not advertised?

3. Terminology and Conventions

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

In addition, this document uses the following terminology:

- o YANG module revision: An instance of a YANG module, with no implied ordering or backwards compatibility between different revisions of the same module."
- o YANG module version: A YANG module revision, but also with an implied partial ordering relationship between other versions of the same module. Each module version must be uniquely identifiable.
- o Non-backwards-compatible (NBC): In the context of this document, the term 'non-backwards-compatible' refers to a change or set of changes between two YANG module revisions that do not adhere to the list of allowable changes specified in Section 11 "Updating a Module" of [RFC7950], with the following additional clarification:
 - * Any addition of, or change to, a "status" statement that allows a server to remove support for a schema node is considered a non-backwards-compatible change

4. The Problem Statement

Considering the issues described in the background, the problem definition can be summarized as follows.

Development of data models for a large collection of communication protocols and system components is difficult and typically only manageable with an iterative development process. Agile development approaches advocate evolutionary development, early delivery, and continual improvement. They are designed to support rapid and flexible response to change. Agile development has been found to be very successful in a world where the objects being modeled undergo constant changes.

The current module versioning scheme relies on the fundamental idea that a definition, once published, never changes its semantics. As a consequence, if a new definition is needed with different non-backwards-compatible semantics, then a new definition must be created

to replace the old definition. The advantage of this versioning scheme is that a definition identified by a module name and a path has fixed semantics that never change. (The details are a bit more nuanced but we simplify things here a bit in order to get the problems worked out clearly.)

There are two main disadvantages of the current YANG versioning scheme:

- o Any non-backwards-compatible change of a definition requires either a new module name or a new path. This has been found costly to support in implementations, in particular on the client side.
- o Since non-backwards-compatible changes require either a new module name or a new path, such changes will impact other modules that import definitions. In fact, with the current module versioning scheme other modules have to opt-in in order to use the new version. This essentially leads to a ripple effect where a non-backwards-compatible change of a core module causes updates on a potentially large number of dependent modules.

Other problems experienced with the current YANG versioning scheme are the following:

- o YANG has a mechanism to mark definitions deprecated but it leaves it open whether implementations are expected to implement deprecated definitions and there is no way (other than trial and error) for a client to find out whether deprecated definitions are supported by a given implementation.
- o YANG does not have a robust mechanism to document which data definitions have changed and to provide guidance how implementations should deal with the change. While it is possible to have this described in general description statements, having these details embedded in general description statements does not make this information accessible to tools.
- o YANG data models often do not exist in isolation and they interact with other software systems or data models that often do allow (controlled) non-backwards-compatible changes. In some cases, YANG models are mechanically derived from other data models that do allow (controlled) non-backwards-compatible changes. In such situations, a robust mapping to YANG requires to have version numbers exposed as part of the module name or a path definition, which has been found to be expensive on the client side (see above).

Given the need to support agile development processes and the disadvantages and problems of the current YANG versioning scheme described above, it is necessary to develop requirements and solutions for a future YANG versioning scheme that better supports agile development processes, whilst retaining the ability for servers to handle clients using older versions of YANG modules.

5. Requirements of a YANG Versioning Solution

The following is a list of requirements that a solution to the problems mentioned above MUST or SHOULD have. The list is grouped by similar requirements but is not presented in a set priority order.

1. Requirements related to making non-backwards-compatible updates to modules:
 - 1.1 A mechanism is REQUIRED to update a module in a non-backwards-compatible way without forcing all modules with import dependencies on the updated module from being updated at the same time (e.g. to change its import to use a new module name).
 - 1.2 Non-backwards-compatible updates of a module MUST not impact clients that only access data nodes of the module that have either not been updated or have been updated in backwards-compatible ways.
 - 1.3 A refined form of YANG's 'import' statement MUST be provided that is more restrictive than "import any revision" and less restrictive than "import a specific revision". Once non-backwards-compatible changes to modules are allowed, the refined import statement is used to express the correct dependency between modules.
 - 1.4 The solution MUST be able to express when non-backwards-compatible changes have occurred between two revisions of a given YANG module.
2. Requirements related to identifying changes between different module revisions:
 - 2.1 Readers of modules, and tools that use modules, MUST be able to determine whether changes between two revisions of a module constitute a backwards-compatible or non-backwards-compatible version change. In addition, it MAY be helpful to identify whether changes represent bug fixes, new functionality, or both.

- 2.2 A mechanism SHOULD be defined to determine whether data nodes between two arbitrary YANG module revisions have (i) not changed, (ii) changed in a backwards-compatible way, (iii) changed in a non-backwards-compatible way.
3. Requirements related to supporting existing clients in a backwards-compatible way:
 - 3.1 The solution MUST provide a mechanism to allow servers to support existing clients in a backwards-compatible way.
 - 3.2 The solution MUST provide a mechanism to support clients that expect an older version of a given module when the current version has had non-backwards-compatible changes.
 - 3.3 Clients are expected to be able to handle unexpected instance data resulting from backwards-compatible changes.
4. Requirements related to managing and documenting the life cycle of data nodes:
 - 4.1 A mechanism is REQUIRED to allow a client to determine whether deprecated nodes are implemented by the server.
 - 4.2 If a data node is deprecated or obsolete then it MUST be possible to document in the YANG module what alternatives exist, the reason for the status change, or any other status related information.
 - 4.3 A mechanism is REQUIRED to indicate that certain definitions in a YANG module will become status obsolete in future revisions but definitions marked as such MUST still be implemented by compliant servers.
5. Requirements related to documentation and education:
 - 5.1 The solution MUST provide guidance to model authors and clients on how to use the new YANG versioning scheme.
 - 5.2 The solution is REQUIRED to describe how to transition from the existing YANG 1.0/1.1 versioning scheme to the new scheme.
 - 5.3 The solution MUST describe how the versioning scheme affects the interpretation of instance data and references to instance data, for which the schema definition has been updated in a non-backwards-compatible way.

6. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following people are members of that design team and have contributed to defining the problem and specifying the requirements:

- o Balazs Lengyel
- o Benoit Claise
- o Ebben Aries
- o Jason Sterne
- o Joe Clarke
- o Juergen Schoenwaelder
- o Mahesh Jethanandani
- o Michael (Wangzitao)
- o Qin Wu
- o Reshad Rahman
- o Rob Wilton

7. Acknowledgments

The design team would like to thank Christian Hopps and Vladimir Vassilev for their feedback and perspectives in shaping and fine tuning the versioning requirements.

One of the inspirations for solving the YANG module versioning comes from OpenConfig. The authors would like to thank Anees Shaikh and Rob Shakir for their helpful input.

8. Security Considerations

The document does not define any new protocol or data model. There is no security impact.

9. IANA Considerations

None

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, <<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>>.

10.2. Informative References

- [RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <<https://www.rfc-editor.org/info/rfc8049>>.
- [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>>.
- [RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <<https://www.rfc-editor.org/info/rfc8299>>.

Author's Address

Joe Clarke (editor)
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2019

R. Wilton
Cisco Systems, Inc.
March 10, 2019

YANG Packages
draft-rwilton-netmod-yang-packages-01

Abstract

This document defines YANG packages, an organizational structure holding a set of related YANG modules, that can be used to simplify the conformance and sharing of YANG schema. It describes how YANG instance data documents are used to define YANG packages, and how the YANG library information published by a server can be augmented with additional packaging related information.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 September 11, 2019.

Copyright Notice

Copyright (c) 2019 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. Terminology and Conventions	2
2. Introduction	3
3. Background on YANG packaging	4
4. Possible alternative solutions	4
4.1. Using module tags	4
4.2. Using YANG library	5
5. Objectives	6
6. Package description	7
6.1. Package definition rules	7
6.2. Package versioning	8
6.3. Client server package conformance	10
6.4. Schema referential completeness	10
6.5. Submodules packaging considerations	11
6.6. Revision history	11
6.7. Uniqueness of packages and global registry	11
7. YANG Packaging instance data	12
8. YANG Packaging additions to YANG library	13
8.1. Package List	14
8.2. Binding from schema to package	14
8.3. Tree diagram	15
9. YANG Packaging groupings	15
10. YANG Modules	17
11. Security Considerations	29
12. IANA Considerations	30
13. Open Questions/Issues	31
14. Acknowledgements	31
15. References	31
15.1. Normative References	31
15.2. Informative References	32
Appendix A. Tree output for ietf-yang-library with package augmentations	32
Appendix B. Examples	34
B.1. Example IETF Network Device YANG package	35
B.2. Example IETF Basic Routing YANG package	37
B.3. Package import conflict resolution example	40
Author's Address	43

1. Terminology and Conventions

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.

This document uses terminology introduced in the YANG versioning requirements draft [I-D.verdt-netmod-yang-versioning-reqs].

This document also makes of the following terminology introduced in the Network Management Datastore Architecture [RFC8342]:

- o datastore schema

In addition, this document makes use of the following terminology:

- o bc: Used as an abbreviation for a backwards-compatible change.
- o nbc: Used as an abbreviation for a non-backwards-compatible change.
- o editorial change: A backwards-compatible change that does not change the YANG module semantics in any way.

Note - the bc/nbc/editorial terminology should probably be defined and referenced from the YANG module versioning solution draft.

2. Introduction

This document defines and describes the YANG [RFC7950] constructs that are used to define and use YANG packages.

A YANG package is an organizational structure that groups a set of related YANG modules together into a consistent versioned definition. YANG packages can themselves refer to and reuse other package definitions.

The draft consists of the following significant sections:

A background section that describes some of the prior work in this area, both within IETF and the wider industry.

An overview of the objectives for a YANG packaging solution, and also what work is out of scope for this document.

The definition of YANG packages, how package definitions are constructed, and how they are used.

How YANG instance data documents [I-D.ietf-netmod-yang-instance-file-format] are used to define particular YANG package instances.

Augmentations to the YANG library [I-D.ietf-netconf-rfc7895bis] content published by servers to include YANG packaging related information.

YANG modules the provide the definitions for YANG packages.

Non-normative examples of YANG package instances are provided in the appendicies.

3. Background on YANG packaging

It has long been acknowledged within the IETF NETMOD community that network management using YANG requires a unit of organization and conformance that is broader in scope than individual YANG modules.

'The YANG Package Statement' [I-D.bierman-netmod-yang-package] proposed a YANG package mechanism based on new YANG language statements, where a YANG package is defined in a file similar to how YANG modules are defined, and would require enhancements to YANG compilers to understand the new statements used to define particular package instances. This document did not progress in the working group, although this may have been due to other higher priority concerns or resource constraints within the working group rather than due to consideration of the technical merits of the proposed approach.

OpenConfig [openconfigsemver] describes an approach to versioning 'bundle releases' based on git tags. I.e. a set of modules, at particular versions, can be marked with the same release tag to indicate that they are known to interoperate together.

The NETMOD WG in general, and the YANG versioning design team in particular, are exploring solutions to the YANG versioning requirements, [I-D.verdt-netmod-yang-versioning-reqs]. Solutions to the versioning requirements can be split into several distinct areas. One draft, TBD (draft-verdt-netmod-yang-semver), has a primary focus on YANG versioning scoped to individual modules. But an overall solution should also consider YANG versioning and conformance scoped to a server's datastore schema. YANG packages may help form part of the solution for versioning at the datastore schema level.

4. Possible alternative solutions

4.1. Using module tags

Module tags have been suggested as an alternative solution, and indeed that can address some of the same requirements as YANG packages but not all of them.

Module tags can be used to group or organize YANG modules. However, this raises the question of where this tag information is stored. Module tags either require that the YANG module files themselves are updated with the module tag information (creating another versioning problem), or for the module tag information to be hosted elsewhere, perhaps in a centralized YANG Catalog, or in instance data documents similar to how YANG packages have been defined in this draft.

One of the principle aims of YANG packages is to be a versioned object that defines a precise set of YANG modules versions that work together. Module tags cannot meet this aim without an explosion of module tags definitions (i.e. a separate module tag must be defined for each package version).

Module tags cannot support the hierarchical scheme to construct YANG schema that is proposed in this draft.

4.2. Using YANG library

Another question is whether it is necessary to define new YANG modules to define YANG packages, and whether YANG library could just be reused in an instance data document. The use of YANG packages is intended to offer several benefits over just using YANG library:

1. Packages allow schema to be built in a hierarchical fashion. [I-D.ietf-netconf-rfc7895bis] only allows one layer of hierarchy (using module sets), and there can be no conflicts between module revisions in different module-sets.
2. Packages can be made available off the box, with a well defined unique name, avoiding the need for clients to download, and construct/check the entire YANG schema for each device, instead they can rely on the named packages. YANG libraries use of checksums are unique only to the device that generated them.
3. Packages are versioned using a semantic versioning scheme, YANG library does not have a schema level semantic version number, although this could potentially be added if required.
4. For a YANG library instance data document to contain the necessary information, it needs both YANG library, and probably also various augmentations (e.g. to include each module's semantic version number), unless a new version of YANG library is defined containing this information. A module definition for a YANG package could be defined to contain all of the necessary information to solve the problem.

5. YANG library is designed to publish information about the modules, datastores, and datastore schema used by a server. It is unclear whether exactly the same information is required for an offline schema definition, or whether these definitions might deviate from each other over time. E.g., some thought needs to be given concerning the relationship between datastores and schema.

5. Objectives

The main goals of YANG package definitions include, but are not restricted to:

- o To act as a simplified YANG conformance mechanism. Rather than conformance being performed against a set of individual YANG module revisions, conformance could also be more simply stated in terms of YANG packages, with a set modifications (e.g. additional modules, deviations, or features).
- o To allow YANG datastore schema to be specified in a more concise way rather than having to list all modules and revisions. YANG package definitions can be defined in documents that can be referenced by a URL rather than requiring explicit lists of modules to be shared between client and server. Hence, a YANG package must contain sufficient information to allow a client or server to precisely construct the schema associated with the package.
- o To provide generic packaging related YANG grouping definitions for use in other YANG modules, as required.
- o To define a mainly linear versioned history of sets of modules versions that are known to work together. I.e. to help mitigate the problem were a client must manage devices from multiple vendors, and vendor A implements version 1.0.0 of module foo and version 2.0.0 of module bar, and vendor B implements version 2.0.0 of module foo and version 1.0.0 of module bar. For a client, trying to interoperate with multiple vendors, and many YANG modules, then finding a consistent lowest common denominator set of YANG module versions may be difficult, or impossible.

Protocol mechanisms of how clients could negotiate which packages or package versions are be used for client server communications are outside the scope of this document. However, the design of the YANG library augmentations for YANG packages are intended to keep open the possibility of such extensions in future work.

Finally, the package definitions proposed by this document are intended to be relatively basic in their definition and the functionality that they support. As industry gains experience using YANG packages, the standard YANG mechanisms of updating, or augmenting, YANG modules could be used to extend the functionality supported by YANG packages.

6. Package description

This document specifies an approach to defining YANG packages that is different to either of the approaches described in the background.

The approach defined here is for a YANG package definition structure to be defined using existing YANG language statements without requiring extensions or new YANG statements. By making use of this structure, particular YANG package instances can be defined as YANG instance data documents [I-D.ietf-netmod-yang-instance-file-format] with well defined names and locations.

The YANG semantic versioning scheme, described in draft-verdt-netmod-yang-semver (TBD), is used to version YANG packages using an equivalent scheme to how individual YANG modules version numbers are changed.

YANG library is augmented to allow servers to report the packages that they implement and to associate those packages back to particular datastore schema.

TODO - It would be helpful if the YANG instance data file format [I-D.ietf-netmod-yang-instance-file-format] could also reference a YANG packages to specify the schema associated with an instance data document. This could either be defined in instance-file-format draft, or as a YANG augmentation as part of this draft.

Each version of a YANG package defines: a set of YANG modules that are implemented at particular versions or revisions; a set of YANG modules that are import-only with particular versions or revisions; and a set of mandatory module features that implementations of the package MUST implement or otherwise deviate.

6.1. Package definition rules

The following rules define how packages are defined:

Every YANG package definition MUST be referentially complete. I.e. all import and include statements for all YANG modules included in a package MUST resolve to a module specified in the package itself, or an imported package.

For a given package, each separate instance of the package MUST have a unique version number that follows the semantic versioning rules described in Section 6.2.

A package MAY have a revision-date. Any package revision-dates MUST be unique for different package versions.

For each module implemented by a package, only a single revision/version MUST be implemented.

The version/revision of a module listed in the package module list supercedes any version/revision of the module listed in a imported package module list. This allows a package to resolve any conflicting implemented module versions/revisions in imported packages.

The replaces-revision leaf-list in the import-only-module list can be used to exclude duplicate revisions of import-only modules from imported packages. Otherwise, the import-only-modules for a package are the import-only-modules from all imported packages combined with any modules listed in the packages import-only-module list.

Modules referenced by a package SHOULD specify the version of the module, both in the package definition and within the module definition itself.

Modules referenced by a package MUST specify the revision date of the module, both in the package definition and within the module definition itself.

6.2. Package versioning

Every YANG package must specify a YANG semantic version field that defines the particular version of the package.

The rules for incrementing the YANG package version number are equivalent to the semantic versioning rules used to version individual YANG modules, defined in TBD (draft-verdt-netmod-yang-semver).

The semantic versioning rules, as they apply to YANG packages, are defined using the following two step process:

The first step is to determined whether the change to the YANG package is classified as a major, minor, or editorial based on the content that has changed in the package relative to the previous version. Where available, the semantic version number of the

referenced elements in the package (imported packages or modules) can be used to help determine what type of change is being made. The formal rules are:

If any of the referenced elements of the package (imported packages or modules) are changed in an nbc way, or if any imported package, module, or mandatory-feature is removed from the package definition, then the package has been updated in an nbc way.

If none of the referenced elements of the package (imported packages, modules) are removed or changed in a nbc way, but some referenced elements are changed in a bc way, or new referenced elements or mandatory-features added, then the package is deemed to be updated in a bc way.

If none of the referenced elements of the package (imported packages, modules) are added, removed, or changed in a nbc or bc way, but some referenced elements have editorial changes then the package is deemed to be updated in an editorial way.

The second step, after it has been determined what type of version change is being made to the YANG package, is for the YANG semantic versioning rules to be applied to update the YANG package semantic version number. The formal rules are:

If the package is being updated in a nbc way, then the package version "X.Y.Z[m|M]" SHOULD be updated to "X+1.0.0" unless that package version has already been defined with different content, in which case the package version "X.Y.Z+1M" MUST be used instead.

If the package is being updated in a bc way, then the package version "X.Y.Z[m|M]" SHOULD be updated to "X.Y+1.0" unless that package version has already been defined with different content, in which case if the current package version is "X.Y.ZM" then it MUST be updated to "X.Y.Z+1M", or otherwise "X.Y.Z+1m".

If the package is being updated in an editorial way, then the package version "X.Y.Z[m|M]" MUST be updated to "X.Y.Z+1[m|M]", retaining the 'm|M' character if it is already present in the previous version."

Package YANG semantic version numbers beginning with 0, i.e "0.X.Y" are regarded as beta definitions and need not follow the nbc rules, and the minor version number can be incremented instead.

In all cases, the 3 number fields that comprise a YANG semantic version number associated with a YANG package MUST uniquely identify the contents of that YANG package.

6.3. Client server package conformance

The YANG semantic versioning scheme used for YANG packages means that a client can determine the nature of changes between two package revisions.

This means that a client is not restricted to working only with servers that advertise exactly the same version of package in YANG library. Instead, reasonable clients should be able to interoperate with a server that supports a package version that is backwards compatible to what the client is designed for.

For example, a client coded to support 'foo' package at version 1.0.0 should interoperate with a server implementing 'foo' package at version 1.3.5, because the YANG semantic versioning rules require that package version 1.3.5 is backwards compatible to version 1.0.0.

This also has a relevance on servers that are capable of supporting version selection because they need not necessarily support every version of a YANG package to ensure good client compatibility. Choosing suitable minor versions within each major version number should generally be sufficient, particular if they can avoid NBC patch level changes (i.e. 'M' labelled versions).

6.4. Schema referential completeness

A YANG package may represent a schema that is 'referentially complete', or 'referentially incomplete'.

If all import statements in all YANG modules included in the package (either directly, or through imported packages) can be resolved to a module revision defined with the YANG package definition, then the package is classified as referentially complete. Conversely, if one or more import statements cannot be resolved to a module specified as part of the package definition, then the package is classified as referentially incomplete.

A package that represents the exact contents of a datastore schema MUST always be referentially complete.

Referentially incomplete packages can be used to group sets of logically related modules together, but without requiring a fixed dependency on all imported 'types' modules, instead leaving the choice of specific revisions of 'types' modules to be resolved when the package definition is used.

6.5. Submodules packaging considerations

As defined in [RFC7950] and draft-verdt-netmod-yang-semver (TBD), YANG conformance and versioning is specified in terms of particular revisions of YANG modules rather than for individual submodules.

However, YANG package definitions also include the list of submodules included by a module, primarily to provide a location of where the submodule definition can be obtained from, allowing a YANG schema to be fully constructed from a YANG package instance-data definition.

Restructuring how a module uses, or does not use, submodules is treated as an editorial level change in YANG semantic versioning, on the condition that there is no change in the modules semantic behavior due to the restructuring.

To ensure that a module and any constituent submodule are tightly related, all 'include' statements in a YANG module SHOULD specify revision-dates of the included submodules. If 'include' statement revision-dates are included in the YANG module then they MUST match the 'revision' field specified for the submodule in the packages's submodules lists.

6.6. Revision history

YANG packages do not contain a revision history, because a linear revision history does not work well for a versioning object that supports branching. In addition, some packages could have frequent revisions, and a long revision history would bloat the package definition.

To mitigate this, the package definition includes a 'previous-version' leaf that indicates the specific version this package definition is based on. By recursively examining the 'previous-version' leaf of a package definition, a full revision history can be dynamically constructed if required.

6.7. Uniqueness of packages and global registry

The name given to a package SHOULD be globally unique, and it SHOULD include an appropriate organization prefix in the name, equivalent to YANG module naming conventions.

Ideally a YANG instance data document defining a particular package version would be publically available at one or more URLs.

7. YANG Packaging instance data

YANG packages are expected to be defined as YANG instance data documents [I-D.ietf-netmod-yang-instance-file-format] using the YANG schema below to define the package data itself.

The instance data document for each version of a YANG package SHOULD be made available at one of more locations accessible via URLs. If one of the listed locations defines a definitive reference implementation for the package definition then it MUST be listed as the first entry in the list.

The "ietf-yang-package" YANG module has the following structure:

```

module: ietf-yang-package
  +--ro yang-package
    +--ro name                yang:yang-identifier
    +--ro version              yang-sem-ver
    +--ro revision-date?      yanglib:revision-identifier
    +--ro location*            inet:uri
    +--ro description?         string
    +--ro reference?           string
    +--ro previous-version?    yang-sem-ver
    +--ro tag*                  tags:tag
    +--ro referentially-complete? boolean
    +--ro mandatory-feature*   string
    +--ro imported-packages* [name version]
      +--ro name                yang:yang-identifier
      +--ro version              yang-sem-ver
      +--ro deviated?           boolean
      +--ro location*            inet:uri
    +--ro module* [name]
      +--ro name                yang:yang-identifier
      +--ro revision?           revision-identifier
      +--ro version?            yang-sem-ver
      +--ro namespace           inet:uri
      +--ro location*            inet:uri
      +--ro submodule* [name]
        +--ro name                yang:yang-identifier
        +--ro revision            yanglib:revision-identifier
        +--ro location*            inet:uri
    +--ro import-only-module* [name revision]
      +--ro name                yang:yang-identifier
      +--ro revision              union
      +--ro version?            yang-sem-ver
      +--ro namespace           inet:uri
      +--ro location*            inet:uri
      +--ro submodule* [name]
        +--ro name                yang:yang-identifier
        +--ro revision            yanglib:revision-identifier
        +--ro location*            inet:uri
    +--ro replaces-revision*   yanglib:revision-identifier

```

8. YANG Packaging additions to YANG library

8.1. Package List

The main addition is a top level 'yang-library/package' list that lists all package of all versions known to the server. Each package itself is defined using imported packages and module-sets to define the specific set of modules implemented and imported by the package. The use of module-sets allows the module definitions to be shared with the existing YANG library schema definitions. The existing rule of RFC 7995bis related to combining modules-sets also applies here, i.e. The combined set of modules defined by the module-sets MUST NOT contain modules implemented at different revisions. I.e. the module-sets leaf-list is directly equivalent to the explicit module and import-only-module lists in the instance data YANG package definition.

The 'yang-library/package' list MAY include multiple versions of a particular package. E.g. if the server is capable of allowing clients to select which package versions should be used by the server.

8.2. Binding from schema to package

The second augmentation is to allow a server to optionally indicate that a schema definition directly relates to a package. Since YANG packages are available offline, it may be sufficient for a client to only check that a compatible version of the YANG package is being implemented by the server without fetching and comparing the full module list.

If a server indicates that its schema maps to a particular package then it MUST support all mandatory-features defined as part of that package, and it MUST NOT have any deviations to the modules defined by the package. A server MAY implement features not specified in the package's mandatory-features list.

If a server cannot faithfully implement a package then it can define a new package to accurately report what it does implement. The new package can include the original package as an imported package, and the new package can define additional modules containing deviations to the original package, allowing the new package to accurately describe the server behavior. There is no specific mechanism provided to indicate that a mandatory-feature is not supported on a server, but deviations MAY be used to disable functionality predicated by a mandatory-feature.

8.3. Tree diagram

The "ietf-yang-library-packages" YANG module has the following structure:

```

module: ietf-yang-library-packages
  augment /yanglib:yang-library:
    +--ro package* [name version]
      +--ro name          yang:yang-identifier
      +--ro version       yang-sem-ver
      +--ro revision-date? yanglib:revision-identifier
      +--ro location*     inet:uri
      +--ro description?  string
      +--ro reference?    string
      +--ro previous-version? yang-sem-ver
      +--ro tag*          tags:tag
      +--ro referentially-complete? boolean
      +--ro mandatory-feature* string
      +--ro imported-packages* [name version]
        +--ro name          yang:yang-identifier
        +--ro version       yang-sem-ver
        +--ro deviated?    boolean
      +--ro module-set*
        -> /yanglib:yang-library/module-set/name
    augment /yanglib:yang-library/yanglib:schema:
      +--ro package
        +--ro name?      -> /yanglib:yang-library/package/name
        +--ro version?   leafref
    augment /yanglib:yang-library/yanglib:module-set/
      yanglib:import-only-module:
      +--ro replaces-revision* yanglib:revision-identifier

```

9. YANG Packaging groupings

Groupings for YANG packaging related constructs are provided in a 'types' module for use by the instance-data and YANG library constructs described previously. They are also available to be used by other modules that have a need for packaging information.

The "ietf-yang-package-types" YANG module has the following structure:

```

module: ietf-yang-package-types

  grouping yang-pkg-identification-leafs

```



```

+----- name          yang:yang-identifier
+----- version       yang-sem-ver
grouping yang-pkg-common-leafs
+----- revision-date? yanglib:revision-identifier
+----- location*      inet:uri
+----- description?   string
+----- reference?     string
+----- previous-version? yang-sem-ver
+----- tag*           tags:tag
+----- referentially-complete? boolean
+----- mandatory-feature* string
+----- imported-packages* [name version]
|   +----- name          yang:yang-identifier
|   +----- version       yang-sem-ver
|   +----- deviated?     boolean
grouping yang-pkg-library-definition
+----- name          yang:yang-identifier
+----- version       yang-sem-ver
+----- revision-date? yanglib:revision-identifier
+----- location*      inet:uri
+----- description?   string
+----- reference?     string
+----- previous-version? yang-sem-ver
+----- tag*           tags:tag
+----- referentially-complete? boolean
+----- mandatory-feature* string
+----- imported-packages* [name version]
|   +----- name          yang:yang-identifier
|   +----- version       yang-sem-ver
|   +----- deviated?     boolean
+----- module-set*
|   -> /yanglib:yang-library/module-set/name
grouping yang-pkg-file-definition
+----- name          yang:yang-identifier
+----- version       yang-sem-ver
+----- revision-date? yanglib:revision-identifier
+----- location*      inet:uri
+----- description?   string
+----- reference?     string
+----- previous-version? yang-sem-ver
+----- tag*           tags:tag
+----- referentially-complete? boolean
+----- mandatory-feature* string
+----- imported-packages* [name version]
|   +----- name          yang:yang-identifier
|   +----- version       yang-sem-ver
|   +----- deviated?     boolean
|   +----- location*      inet:uri

```

```

+---- module* [name]
|   +---- name      yang:yang-identifier
|   +---- revision? revision-identifier
|   +---- version?  yang-sem-ver
|   +---- namespace inet:uri
|   +---- location* inet:uri
|   +---- submodule* [name]
|       +---- name?      yang:yang-identifier
|       +---- revision  yanglib:revision-identifier
|       +---- location* inet:uri
+---- import-only-module* [name revision]
    +---- name?      yang:yang-identifier
    +---- revision?  union
    +---- version?   yang-sem-ver
    +---- namespace  inet:uri
    +---- location*  inet:uri
    +---- submodule* [name]
        +---- name?      yang:yang-identifier
        +---- revision  yanglib:revision-identifier
        +---- location*  inet:uri
+---- replaces-revision* yanglib:revision-identifier

```

10. YANG Modules

The YANG module definitions for the modules described in the previous sections.

```

<CODE BEGINS> file "ietf-yang-package-types@2018-11-26.yang"
module ietf-yang-package-types {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-package-types";
  prefix "pkg-types";

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference "RFC 7895bis: YANG Library";
  }
  import ietf-module-tags {

```

```
    prefix tags;
    reference "XXX, (draft-ietf-netmod-module-tags-03): YANG Module Tags";
}

organization
  "IETF NETMOD (Network Modeling) Working Group";

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

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

description
  "This module provides type and grouping definitions for YANG
  packages.

  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; see
  the RFC itself for full legal notices."

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.
revision 2018-11-26 {
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Schema Versioning.";
}

/*
 * Typedefs
 */

typedef yang-sem-ver {
  type string {
```

```
    pattern '\d+[.]\d+[.]\d+[mM]?';
  }
  description
    "Represents a YANG semantic version number.";
  reference
    "TODO - Should be defined by YANG versioning types module";
}

/*
 * Groupings
 */

grouping yang-pkg-identification-leafs {
  description
    "Parameters for identifying a specific version of a YANG
    package";

  leaf name {
    type yang:yang-identifier;
    mandatory true;
    description
      "The YANG package name.";
  }

  leaf version {
    type yang-sem-ver;
    mandatory true;
    description
      "YANG package version. Follows YANG semantic versions rules
      defined in XXX";
  }
}

grouping yang-pkg-common-leafs {
  description
    "Defines definitions common to all YANG package definitions.";

  leaf revision-date {
    type yanglib:revision-identifier;

    description
      "An optional revision identifier of when this package version
      was created. This does not need to be unique across all
      versions of a package.";
  }

  leaf-list location {
    type inet:uri;
  }
}
```

```
description
  "Contains a URL that represents where an instance data file
   for this YANG package can be found.

  This leaf will only be present if there is a URL
  available for retrieval of the schema for this entry.

  If multiple locations are provided, then the first location
  in the leaf-list MUST be the definitive location that
  uniquely identifies this package";
}

leaf description {
  type string;

  description "Provides a description of the package";
}

leaf reference {
  type string;

  description "Allows for a reference for the package";
}

leaf previous-version {
  type yang-sem-ver;
  description
    "The previous package version that this version has been
     derived from. This leaf allows a full version history graph
     to be constructed if required.";
}

leaf-list tag {
  type tags:tag;
  description
    "Tags associated with a YANG package. Module tags defined in
     XXX, ietf-netmod-module-tags can be used here but with the
     modification that the tag applies to the entire package
     rather than a specific module. See the IANA 'YANG Module Tag
     Prefix' registry for reserved prefixes and the IANA 'YANG
     Module IETF Tag' registry for IETF standard tags.";
}

leaf referentially-complete {
  type boolean;
  default true;
  description
    "Indicates whether the schema defined by this package is
```

```
referentially complete. I.e. all module imports can be
resolved to a module explicitly defined in this package or one
of the imported packages.";
}

leaf-list mandatory-feature {
  type string;
  // TODO - Is there a better type for this?
  description
    "List all features from modules included in the package that
    MUST be supported by any server implementing the package.
    All other features defined in included packages are OPTIONAL
    to implement.

    Features are identified using <module-name>:<feature>";
}

list imported-packages {
  key "name version";
  description
    "An entry in this list represents a package that is imported
    as part of the package definition.

    If packages implement different revisions or versions of the
    same module, then an explicit entry in the module list MUST
    be provided to select the specific module version
    'implemented' by this package definition.

    For import-only modules, the replaces-revision leaf-list can
    be used to select the specific module versions imported by
    this package.";
  reference
    "XXX";

  uses yang-pkg-identification-leafs;

  leaf deviated {
    type boolean;
    default true;
    description
      "Set to true if any data nodes in this package are modified
      in a non backwards compatible way, either through the use
      of deviations, or because one of the modules has been
      replaced by an earlier module version.";
  }
}
}
```

```
grouping yang-pkg-file-definition {
  description
    "The set of parameters that describe a particular YANG package.";

  uses yang-pkg-identification-leafs;

  uses yang-pkg-common-leafs {
    augment "imported-packages" {
      description "Add the package location path";

      leaf-list location {
        type inet:uri;
        description
          "Contains a URL that represents where an instance data
           file for this YANG package can be found.

          This leaf will only be present if there is a URL
          available for retrieval of the schema for this entry.

          If multiple locations are provided, then the first
          location in the leaf-list MUST be the definitive location
          that uniquely identifies this package";
      }
    }
  }
}

list module {
  key "name";
  description
    "An entry in this list represents a module that must be
     implemented by a server implementing this package, as per
     RFC 7950 section 5.6.5, with a particular set of supported
     features and deviations.

     A entry in this list overrides any module version
     'implemented' by an imported package";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";

  uses yanglib:module-identification-leafs;

  leaf version {
    type yang-sem-ver;
    description
      "The YANG module or submodule version.  If no version
       statement is present in the YANG module or submodule, this
       leaf is not instantiated.";
  }
}
```

```
leaf namespace {
  type inet:uri;
  mandatory true;
  description
    "The XML namespace identifier for this module.";
}
uses yanglib:location-leaf-list;

list submodule {
  key "name";
  description
    "Each entry represents one submodule within the
    parent module.";

  leaf name {
    type yang:yang-identifier;
    description
      "The YANG submodule name.";
  }
  leaf revision {
    type yanglib:revision-identifier;
    mandatory true;
    description
      "The YANG submodule revision date.  If the parent module
      include statement for this submodule includes a revision
      date then it MUST match this leaf's value.";
  }

  uses yanglib:location-leaf-list;
}

list import-only-module {
  key "name revision";
  description
    "An entry in this list indicates that the server imports
    reusable definitions from the specified revision of the
    module, but does not implement any protocol accessible
    objects from this revision.

    Multiple entries for the same module name MAY exist.  This
    can occur if multiple modules import the same module, but
    specify different revision-dates in the import statements.";

  leaf name {
    type yang:yang-identifier;
    description
      "The YANG module name.";
```



```
}
leaf revision {
  type union {
    type yanglib:revision-identifier;
    type string {
      length 0;
    }
  }
  description
    "The YANG module revision date. A zero-length string is
    used if no revision statement is present in the YANG
    module.";
}
leaf version {
  type yang-sem-ver;
  description
    "The YANG module or submodule version. If no version
    statement is present in the YANG module or submodule, this
    leaf is not instantiated.";
}
leaf namespace {
  type inet:uri;
  mandatory true;
  description
    "The XML namespace identifier for this module.";
}

uses yanglib:location-leaf-list;

list submodule {
  key "name";
  description
    "Each entry represents one submodule within the
    parent module.";

  leaf name {
    type yang:yang-identifier;
    description
      "The YANG submodule name.";
  }
  leaf revision {
    type yanglib:revision-identifier;
    mandatory true;
    description
      "The YANG submodule revision date. If the parent module
      include statement for this submodule includes a revision
      date then it MUST match this leaf's value.";
  }
}
```

```
    uses yanglib:location-leaf-list;
  }

  leaf-list replaces-revision {
    type yanglib:revision-identifier;
    description
      "Gives the revision of an import-only-module defined in
      an imported package that is replaced by this
      import-only-module revision.";
  }
}

grouping yang-pkg-library-definition {
  description
    "The set of parameters that describe a particular YANG package.";

  uses yang-pkg-identification-leafs;
  uses yang-pkg-common-leafs;

  leaf-list module-set {
    type leafref {
      path "/yanglib:yang-library/yanglib:module-set/yanglib:name";
    }
    description
      "Describes any modules in addition to, and replacing, and
      modules defined in the imported packages.

      If a non import-only module appears in multiple module sets,
      then the module revision and the associated features and
      deviations must be identical.";
  }
}
}
}
<CODE ENDS>
```

```
<CODE BEGINS> file "ietf-yang-package2018-11-26.yang"
module ietf-yang-package {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-package";
  prefix pkg;

  import ietf-yang-package-types {
    prefix pkg-types;
    reference "RFC XXX: YANG Schema Versioning.";
  }
}
```

organization

"IETF NETMOD (Network Modeling) Working Group";

contact

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

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

description

"This module provides a definition of a YANG package, which is used as the schema for an YANG instance data document specifying a YANG package.

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; see the RFC itself for full legal notices.";

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.

```
revision 2018-11-26 {  
  description  
    "Initial revision";  
  reference  
    "RFC XXXX: YANG Schema Versioning."  
}
```

```
/*  
 * Top-level container  
 */
```

```
container yang-package {  
  config false;  
  description  
    "Defines a YANG package.
```

Intended to be used to specify a YANG package as an instance data document.";

```
    uses pkg-types:yang-pkg-file-definition;
  }
}
<CODE ENDS>
```

```
<CODE BEGINS> file "ietf-yang-library-packages@2018-11-26.yang"
module ietf-yang-library-packages {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-library-packages";
  prefix pkg;

  import ietf-yang-package-types {
    prefix pkg-types;
    reference "RFC XXX: YANG Packages.";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference "RFC 7895bis: YANG Library";
  }

  organization
    "IETF NETMOD (Network Modeling) Working Group";

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

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

  description
    "This module provides defined augmentations to YANG library to
    allow a server to report YANG package information.

    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; see the RFC itself for full legal notices.";

```
// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.
revision 2018-11-26 {
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Schema Versioning.";
}

/*
 * Add in the list of packaged into YANG library.
 */
augment "/yanglib:yang-library" {
  description "Add YANG package definitions into YANG library";

  list package {
    config "false";
    key "name version";

    description "Defines the packages available on this server.";

    uses "pkg-types:yang-pkg-library-definition";
  }
}

/*
 * Allow schema to be related to a YANG package.
 */
augment "/yanglib:yang-library/yanglib:schema" {
  description
    "Allow datastore schema to be related to a YANG package";

  container package {
    leaf name {
      type leafref {
        path "/yanglib:yang-library/package/name";
      }
      description
        "The name of the package this schema relates to.";
    }
  }
}
```

```
    leaf version {
      type leafref {
        path '/yanglib:yang-library/'
          + 'package[name = current()/../name]/version';
      }

      description
        "The version of the package this schema relates to.";
    }

    description
      "Describes which package the schema directly relates to, if
        any.";
  }
}

/*
 * Allow import-only modules to list the versions that they are
 * replacing.
 */

augment
  "/yanglib:yang-library/yanglib:module-set/" +
  "yanglib:import-only-module" {

    description
      "Add replaces-revision to import-only-module definitions";

    leaf-list replaces-revision {
      type yanglib:revision-identifier;
      description
        "Gives the revision of an import-only-module defined in an
          imported package that is replaced by this import-only-module
          revision.

          Only used for YANG package definitions";
    }
  }
}
<CODE ENDS>
```

11. Security Considerations

The YANG modules specified in this document defines a schema for data that is accessed by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure

transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

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

Similarly to YANG library [I-D.ietf-netconf-rfc7895bis], some of the readable data nodes in these YANG modules 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.

One additional key different to YANG library, is that the 'ietf-yang-package' YANG module defines a schema to allow YANG packages to be defined in YANG instance data documents, that are outside the security controls of the network management protocols. Hence, it is important to also consider controlling access to these package instance data documents to restrict access to sensitive information.

As per the YANG library security considerations, the module, revision and version information in YANG packages may help an attacker identify the server capabilities and server implementations with known bugs since the set of YANG modules supported by a server may reveal the kind of device and the manufacturer of the device. Server vulnerabilities may be specific to particular modules, 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 packaging information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

12. IANA Considerations

It is expected that a central registry of standard YANG package definitions is required to support this packaging solution.

It is unclear whether an IANA registry is also required to manage specific package versions. It is highly desirable to have a specific canonical location, under IETF control, where the definitive YANG package versions can be obtained from.

TODO - Add IANA registrations for YANG modules defined in this draft.

13. Open Questions/Issues

All issues, along with the draft text, are currently being tracked at <https://github.com/rgwilton/YANG-Packages-Draft/issues/>

14. Acknowledgements

Feedback helping shape this document has kindly been provided by Andy Bierman, Ladislav Lhotka, and Jason Sterne.

15. References

15.1. Normative References

- [I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-07 (work in progress), October 2018.
- [I-D.ietf-netmod-module-tags]
Hopps, C., Berger, L., and D. Bogdanovic, "YANG Module Tags", draft-ietf-netmod-module-tags-07 (work in progress), March 2019.
- [I-D.ietf-netmod-yang-instance-file-format]
Lengyel, B. and B. Claise, "YANG Instance Data File Format", draft-ietf-netmod-yang-instance-file-format-02 (work in progress), February 2019.
- [I-D.verdt-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-02 (work in progress), November 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

15.2. Informative References

- [I-D.bierman-netmod-yang-package] Bierman, A., "The YANG Package Statement", draft-bierman-netmod-yang-package-00 (work in progress), July 2015.
- [I-D.ietf-netmod-artwork-folding] Watsen, K., Wu, Q., Farrel, A., and B. Claise, "Handling Long Lines in Artwork in Internet-Drafts and RFCs", draft-ietf-netmod-artwork-folding-00 (work in progress), November 2018.
- [openconfigsemver] "Semantic Versioning for Openconfig Models", <<http://www.openconfig.net/docs/semver/>>.
- [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>>.

Appendix A. Tree output for ietf-yang-library with package augmentations

Complete tree output for ietf-yang-library with package augmentations.

```

module: ietf-yang-library
  +--ro yang-library
    |
    | +--ro module-set* [name]
    | |
    | | +--ro name string
    | | +--ro module* [name]
    | | |
    | | | +--ro name yang:yang-identifier
    | | | +--ro revision? revision-identifier
    | | | +--ro namespace inet:uri
    | | | +--ro location* inet:uri
    | | | +--ro submodule* [name]
    | | | |
    | | | | +--ro name yang:yang-identifier
    | | | | +--ro revision? revision-identifier
    | | | | +--ro location* inet:uri
    | | | +--ro feature* yang:yang-identifier
    | | | +--ro deviation* -> ../../module/name
    | | +--ro import-only-module* [name revision]
    | | |
    | | | +--ro name yang:yang-identifier
    | | | +--ro revision union
    | | | +--ro namespace inet:uri
    | | | +--ro location* inet:uri
    | | | +--ro submodule* [name]
    | | | |
    | | | | +--ro name yang:yang-identifier
    | | | | +--ro revision? revision-identifier
    | | | | +--ro location* inet:uri
    | | | +--ro pkg:replaces-revision*
    | | | | yanglib:revision-identifier
    | +--ro schema* [name]
    | |
    | | +--ro name string
    | | +--ro module-set* -> ../../module-set/name
    | | +--ro pkg:package
    | | | +--ro pkg:name?
    | | | | -> /yanglib:yang-library/package/name
    | | | +--ro pkg:version? leafref
    | +--ro datastore* [name]
    | |
    | | +--ro name ds:datastore-ref
    | | +--ro schema -> ../../schema/name
    | +--ro content-id string
    | +--ro pkg:package* [name version]
    | |
    | | +--ro pkg:name yang:yang-identifier
    | | +--ro pkg:version yang-sem-ver
    | | +--ro pkg:revision-date?
    | | | yanglib:revision-identifier
    | | +--ro pkg:location* inet:uri
    | | +--ro pkg:description? string

```

```

|      +--ro pkg:reference?                string
|      +--ro pkg:previous-version?         yang-sem-ver
|      +--ro pkg:tag*                      tags:tag
|      +--ro pkg:referentially-complete?   boolean
|      +--ro pkg:mandatory-feature*        string
|      +--ro pkg:imported-packages* [name version]
|      |      +--ro pkg:name                yang:yang-identifier
|      |      +--ro pkg:version             yang-sem-ver
|      |      +--ro pkg:deviated?          boolean
|      +--ro pkg:module-set*
|          -> /yanglib:yang-library/module-set/name
x--ro modules-state
  x--ro module-set-id    string
  x--ro module* [name revision]
    x--ro name            yang:yang-identifier
    x--ro revision        union
    +--ro schema?         inet:uri
    x--ro namespace       inet:uri
    x--ro feature*        yang:yang-identifier
    x--ro deviation* [name revision]
      |      x--ro name            yang:yang-identifier
      |      x--ro revision        union
    x--ro conformance-type enumeration
    x--ro submodule* [name revision]
      x--ro name            yang:yang-identifier
      x--ro revision        union
      +--ro schema?         inet:uri

notifications:
  +---n yang-library-update
  |   +--ro content-id    -> /yang-library/content-id
  x---n yang-library-change
    x--ro module-set-id   -> /modules-state/module-set-id

```

Appendix B. Examples

This section provides various examples of YANG packages, and as such this text is non-normative. The purpose of the examples is to only illustrate the file format of YANG packages, and how package dependencies work. It does not imply that such packages will be defined by IETF, or which modules would be included in those packages even if they were defined.

B.1. Example IETF Network Device YANG package

This section provides an instance data document example of an IETF Network Device YANG package formatted in JSON.

This example package is intended to represent the standard set of YANG modules, with import dependencies, to implement a basic network device without any dynamic routing or layer 2 services. E.g., it includes functionality such as system information, interface and basic IP configuration.

As for all YANG packages, all import dependencies are fully resolved. Because this example uses YANG modules that have been standardized before YANG semantic versioning, they modules are referenced by revision date rather than version number.

```
<CODE BEGINS> file "example-ietf-network-device-pkg.json"
===== NOTE: '\\\'' line wrapping per BCP XX (RFC XXXX) =====

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-ietf-network-device-pkg",
    "target-ptr": "TBD",
    "timestamp": "2018-12-13T17:00:00Z",
    "description": "Example IETF network device YANG package definiti\
\ion",
    "content-data": {
      "ietf-yang-package:yang-package": {
        "name": "example-ietf-network-device",
        "version": "1.1.2",
        "namespace": "urn:ietf:params:xml:ns:yang-pkg:ietf-network-d\
\evice",
        "location": "file://example.org/yang/packages/ietf-network-d\
\evice@v1.1.2.json",
        "description": "This package defines a small sample set of Y\
\ANG modules that could represent the basic set of modules that a st\
\andard network device might be expected to support.",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "revision-date": "2018-11-26",
        "module": [
          {
            "name": "iana-crypt-hash",
            "revision": "2014-08-06",
            "namespace": "urn:ietf:params:xml:ns:yang:iana-crypt-has\
\h",
            "location": "https://raw.githubusercontent.com/YangModel\
\s/yang/master/standard/ietf/RFC/iana-crypt-hash%402014-08-06.yang"
```

```

    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-system",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-system%402014-08-06.yang"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2018-02-20",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-interface\
\s",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-interfaces%402018-02-20.yang"
    },
    {
      "name": "ietf-netconf-acm",
      "revision": "2018-02-14",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-netconf-a\
\cm",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-netconf-acm%402018-02-14.yang"
    },
    {
      "name": "ietf-key-chain",
      "revision": "2017-06-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-key-chain\
\s",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-key-chain%402017-06-15.yang"
    },
    {
      "name": "ietf-ip",
      "revision": "2018-02-22",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-ip%402018-02-22.yang"
    }
  ],
  "import-only-module": [
    {
      "name": "ietf-yang-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-type\
\s",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/ietf/master/standard/ietf/RFC/ietf-yang-types%402013-07-15.yang"
    }
  ]
}

```

```

    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-type\
\s",
      "location": "https://raw.githubusercontent.com/YangModel\
\s/yang/master/standard/ietf/RFC/ietf-inet-types%402013-07-15.yang"
    }
  ]
}
}
}
}
}
<CODE ENDS>

```

B.2. Example IETF Basic Routing YANG package

This section provides an instance data document example of a basic IETF Routing YANG package formatted in JSON.

This example package is intended to represent the standard set of YANG modules, with import dependencies, that builds upon the example-ietf-network-device YANG package to add support for basic dynamic routing and ACLs.

As for all YANG packages, all import dependencies are fully resolved. Because this example uses YANG modules that have been standardized before YANG semantic versioning, they modules are referenced by revision date rather than version number. Locations have been excluded where they are not currently known, e.g., for YANG modules defined in IETF drafts. In a normal YANG package, locations would be expected to be provided for all YANG modules.

```

<CODE BEGINS> file "example-ietf-routing-pkg.json"
===== NOTE: '\n' line wrapping per BCP XX (RFC XXXX) =====

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-ietf-routing-pkg",
    "target-ptr": "TBD",
    "timestamp": "2018-12-13T17:00:00Z",
    "description": "Example IETF routing YANG package definition",
    "content-data": {
      "ietf-yang-package:yang-package": {
        "name": "example-ietf-routing",

```

```

    "version": "1.3.1",
    "namespace": "urn:ietf:params:xml:ns:yang-pkg:ietf-routing",
    "location": "file://example.org/yang/packages/ietf-routing@v\
1.3.1.json",
    "description": "This package defines a small sample set of I\
ETF routing YANG modules that could represent the set of IETF routi\
ng functionality that a basic IP network device might be expected t\
o support.",
    "reference": "XXX, draft-rwilton-netmod-yang-packages",
    "revision-date": "2018-11-26",
    "imported-packages": [
      {
        "name": "ietf-network-device",
        "version": "1.1.2",
        "location": [
          "http://example.org/yang/packages/ietf-network-device@v\
1.1.2.json"
        ]
      }
    ],
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2018-03-13",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "location": [
          "https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC/ietf-routing@2018-03-13.yang"
        ]
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2018-03-13",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-ipv4-unca\
st-routing",
        "location": [
          "https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC/ietf-ipv4-unicast-routing@2018-03-13.yang"
        ]
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2018-03-13",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-ipv6-unca\
st-routing",
        "location": [
          "https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC/ietf-ipv6-unicast-routing@2018-03-13.yang"
        ]
      }
    ]
  }
}

```

```

    ]
  },
  {
    "name": "ietf-isis",
    "revision": "2018-12-11",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-isis"
  },
  {
    "name": "ietf-interfaces-common",
    "revision": "2018-07-02",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-interface\
\s-common"
  },
  {
    "name": "ietf-if-l3-vlan",
    "revision": "2017-10-30",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-if-l3-vla\
\n"
  },
  {
    "name": "ietf-routing-policy",
    "revision": "2018-10-19",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing-p\
\olicy"
  },
  {
    "name": "ietf-bgp",
    "revision": "2018-05-09",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-bgp"
  },
  {
    "name": "ietf-access-control-list",
    "revision": "2018-11-06",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-access-co\
\ntrol-list"
  }
],
"import-only-module": [
  {
    "name": "ietf-routing-types",
    "revision": "2017-12-04",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing-t\
ypes",
    "location": [
      "https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC/ietf-routing-types@2017-12-04.yang"
    ]
  }
],

```



```

        {
            "name": "iana-routing-types",
            "revision": "2017-12-04",
            "namespace": "urn:ietf:params:xml:ns:yang:iana-routing-t\
\ypes",
            "location": [
                "https://raw.githubusercontent.com/YangModels/yang/mas\
\ter/standard/ietf/RFC/iana-routing-types@2017-12-04.yang"
            ]
        },
        {
            "name": "ietf-bgp-types",
            "revision": "2018-05-09",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-bgp-types"
        },
        {
            "name": "ietf-packet-fields",
            "revision": "2018-11-06",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-packet-fi\
\elds"
        },
        {
            "name": "ietf-ethertypes",
            "revision": "2018-11-06",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-ethertype\
\s"
        }
    ]
}
}
}
}
<CODE ENDS>

```

B.3. Package import conflict resolution example

This section provides an example of how a package can resolve conflicting module versions from imported packages.

In this example, YANG package 'example-3-pkg' imports both 'example-import-1' and 'example-import-2' packages. However, the two imported packages implement different versions of 'example-module-A' so the 'example-3-pkg' package selects version '1.2.3' to resolve the conflict. Similarly, for import-only modules, the 'example-3-pkg' package does not require both versions of example-types-module-C to be imported, so it indicates that it only imports revision '2018-11-26' and not '2018-01-01'.

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-import-1-pkg",
    "description": "First imported example package",
    "content-data": {
      "ietf-yang-package:yang-package": {
        "name": "example-import-1",
        "version": "1.0.0",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "revision-date": "2018-01-01",
        "module": [
          {
            "name": "example-module-A",
            "version": "1.0.0"
          },
          {
            "name": "example-module-B",
            "version": "1.0.0"
          }
        ],
        "import-only-module": [
          {
            "name": "example-types-module-C",
            "revision": "2018-01-01"
          },
          {
            "name": "example-types-module-D",
            "revision": "2018-01-01"
          }
        ]
      }
    }
  }
}

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-import-2-pkg",
    "description": "Second imported example package",
    "content-data": {
      "ietf-yang-package:yang-package": {
        "name": "example-import-2",
        "version": "2.0.0",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "revision-date": "2018-11-26",
        "module": [
          {
            "name": "example-module-A",
```

```
        "version": "1.2.3"
      },
      {
        "name": "example-module-E",
        "version": "1.1.0"
      }
    ],
    "import-only-module": [
      {
        "name": "example-types-module-C",
        "revision": "2018-11-26"
      },
      {
        "name": "example-types-module-D",
        "revision": "2018-11-26"
      }
    ]
  }
}

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-3-pkg",
    "description": "Importing example package",
    "content-data": {
      "ietf-yang-package:yang-package": {
        "name": "example-3",
        "version": "1.0.0",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "revision-date": "2018-11-26",
        "imported-packages": [
          {
            "name": "example-import-1",
            "version": "1.0.0"
          },
          {
            "name": "example-import-2",
            "version": "2.0.0"
          }
        ],
        "module": [
          {
            "name": "example-module-A",
            "version": "1.2.3"
          }
        ]
      }
    }
  }
}
```

```
    "import-only-module": [  
      {  
        "name": "example-types-module-C",  
        "revision": "2018-11-26",  
        "replaces-revision": [ "2018-01-01 "]  
      }  
    ]  
  }  
}  
}
```

Author's Address

Robert Wilton
Cisco Systems, Inc.

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: January 4, 2020

B. Claise
J. Clarke
R. Rahman
R. Wilton, Ed.
Cisco Systems, Inc.
B. Lengyel
Ericsson
J. Sterne
Nokia
K. D'Souza
AT&T
July 3, 2019

Updated YANG Module Revision Handling
draft-verdt-netmod-yang-module-versioning-00

Abstract

This document specifies a new YANG module update procedure that can document when non-backwards-compatible changes have occurred during the evolution of a YANG module. It extends the YANG import statement with an earliest revision filter to better represent inter-module dependencies. It provides help and guidelines for managing the lifecycle of YANG modules and individual schema nodes. This document updates RFC 7950, RFC 8407 and RFC 8525.

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 4, 2020.

Copyright Notice

Copyright (c) 2019 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. Updates to YANG RFCs	4
2. Terminology and Conventions	4
3. Refinements to YANG revision handling	4
3.1. Updating a YANG module with a new revision	5
3.1.1. Backwards-compatible changes	5
3.1.2. Non-backwards-compatible changes	6
3.2. nbc-changes revision extension statement	6
3.3. Revision label	6
3.4. YANG status description extension statement	7
3.5. Examples for updating the YANG module revision history	7
4. Import by derived revision	9
4.1. Module import examples	10
5. Updates to ietf-yang-library	12
5.1. Advertising revision-label	12
5.2. Resolving ambiguous module imports	12
5.3. Reporting how deprecated and obsolete nodes are handled	13
6. Versioning of YANG instance data	13
7. Guidelines for using the YANG module update rules	14
7.1. Guidelines for YANG module authors	14
7.1.1. Making non-backwards-compatible changes to a YANG module	14
7.2. Versioning Considerations for Clients	16
8. Module Versioning Extension YANG Modules	16
9. Contributors	22
10. Security Considerations	23
11. IANA Considerations	23
11.1. YANG Module Registrations	23
12. References	23
12.1. Normative References	23
12.2. Informative References	24

Appendix A. Appendix	25
A.1. Examples of guidelines for making NBC changes to a YANG module	25
A.1.1. Removing a data node	25
A.1.2. Changing the type of a leaf node	26
A.1.3. Reducing the range of a leaf node	27
A.1.4. Changing the key of a list	27
A.1.5. Renaming a node	28
A.1.6. Changing a default value	29
Authors' Addresses	29

1. Introduction

This document defines a solution to the YANG module lifecycle problems described in [I-D.verdt-netmod-yang-versioning-reqs]. Complementary documents provide a complete solution to the YANG versioning requirements, with the overall relationship of the solution drafts described in [I-D.verdt-netmod-yang-solutions].

Specifically, this document recognises a need (within standards organizations, vendors, and the industry) to sometimes allow YANG modules to evolve with non-backwards-compatible changes, which could cause breakage to clients and importing YANG modules. Accepting that non-backwards-compatible changes do sometimes occur, it is important to have mechanisms to report where these changes occur, and to manage their effect on clients and the broader YANG ecosystem.

The solution comprises five parts:

Refinements to the YANG 1.1 module revision update procedure, supported by new extension statements to indicate when a revision contains non-backwards-compatible changes, and an optional revision label.

A YANG extension statement allowing YANG module imports to specify an earliest module revision that may satisfy the import dependency.

Updates and augmentations to ietf-yang-library to include the revision label in the module descriptions, to report how "deprecated" and "obsolete" nodes are handled by a server, and to clarify how module imports are resolved when multiple versions could otherwise be chosen.

Considerations of how versioning applies to YANG instance data.

Guidelines for how the YANG module update rules defined in this document should be used, along with examples.

Open issues are tracked at <<https://github.com/netmod-wg/yang-ver-dt/issues>>.

1.1. Updates to YANG RFCs

This document updates [RFC7950] section 11. Section 3 describes modifications to YANG revision handling and update rules, and Section 4 describes a YANG extension statement to do import by derived revision.

This document updates [RFC8525] section 3. Section 5 defines how a client of a YANG library datastore schema chooses which revision of an import-only module is used to resolve a module import when the definition is otherwise ambiguous.

This document updates [RFC8407] section 4.7. Section 7 provides guidelines on managing the lifecycle of YANG modules that may contain non-backwards-compatible changes and a branched revision history.

2. Terminology and Conventions

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.

In addition, this document uses the terminology:

- o YANG module revision: An instance of a YANG module, uniquely identified with a revision date, with no implied ordering or backwards compatibility between different revisions of the same module.
- o Backwards-compatible (BC) change: A backwards-compatible change between two YANG module revisions, as defined in Section 3.1.1
- o Non-backwards-compatible (NBC) change: A non-backwards-compatible change between two YANG module revisions, as defined in Section 3.1.2

3. Refinements to YANG revision handling

[RFC7950] assumes, but does not explicitly state, that the revision history for a YANG module is strictly linear, i.e., it is prohibited to have two independent revisions of a YANG module that are both directly derived from the same parent revision.

This document clarifies [RFC7950] to explicitly allow non linear development of YANG module revisions, so modules MAY have multiple revisions that directly derive from the same parent revision. As per [RFC7950], YANG module revisions continue to be uniquely identified by the module's revision date, and hence all revisions of a module MUST have unique revision dates.

A module's name and revision date identifies a specific immutable definition of that module within its revision history. Hence, if a module includes submodules then the module's "include" statements MUST use "revision-date" substatements to specify the exact revision date of each included submodule.

[RFC7950] section 11 requires that all updates to a YANG module are BC to the previous revision of the module. This document allows for more flexible evolution of YANG modules: NBC changes between module revisions are allowed and are documented using a new "nbc-changes" YANG extension statement in the module revision history.

3.1. Updating a YANG module with a new revision

This section updates [RFC7950] section 11 to refine the rules for permissible changes when a new YANG module revision is created.

Where pragmatic, updates to YANG modules SHOULD be backwards-compatible, following the definition in Section 3.1.1.

A new module revision MAY contain NBC changes, i.e., the semantics of an existing definition MAY be changed in an NBC way without requiring a new definition with a new identifier. A new module revision with NBC changes MUST include the "rev:nbc-changes" extension substatement to signal the potential for incompatibility to existing module users and readers.

3.1.1. Backwards-compatible changes

A change between two module revisions is defined as being "backwards-compatible" if the change conforms to the module update rules specified in [RFC7950] section 11, updated by the following rules:

- o A "status" "deprecated" statement MAY be added, or changed from "current" to "deprecated", but adding or changing "status" to "obsolete" is not a backwards-compatible change.
- o Obsolete definitions MAY be removed from published modules, and are classified as backwards-compatible changes. In some circumstances it may be helpful to retain the obsolete definitions

to ensure that their identifiers are not reused with a different meaning.

- o In statements that have any data definition statements as substatements, those data definition substatements MAY be reordered, as long as they do not change the ordering or any "rpc" "input" substatements. If new data definition statements are added, they can be added anywhere in the sequence of existing substatements.

3.1.2. Non-backwards-compatible changes

Any changes to YANG modules that are not defined by Section 3.1.1 as being backwards-compatible are classified as "non-backwards-compatible" changes.

3.2. nbc-changes revision extension statement

The "rev:nbc-changes" extension statement is used to indicate YANG module revisions that contain NBC changes.

If a revision of a YANG module contains changes, relative to the preceding revision in the revision history, that do not conform to the module update rules defined in Section 3.1.1, then a "rev:nbc-changes" extension statement MUST be added as a substatement to the "revision" statement.

Conversely, if a revision does not contain an "rev:nbc-changes" extension substatement then all changes, relative to the preceding revision in the revision history, MUST be backwards-compatible.

3.3. Revision label

Each revision entry in a module or submodule MAY have a revision label associated with it, providing an alternative alias to identify a particular revision of a module or submodule. The revision label could be used to provide an additional versioning identifier associated with the revision. E.g., one option for a versioning scheme that could be used is [TODO - Reference semver draft].

The revision date and revision label within a submodule's revision history have no effect on the including module's revision. Submodules MUST NOT use revision label schemes that could be confused with the including module's revision label scheme.

If a revision has an associated revision label, then it may be used instead of the revision date in two places:

In an "rev:revision-or-derived" extension statement argument.

In the filename of a YANG module, where it takes the form: module-or-submodule-name ['@' revision-label] ('.yang' / '.yin')

3.4. YANG status description extension statement

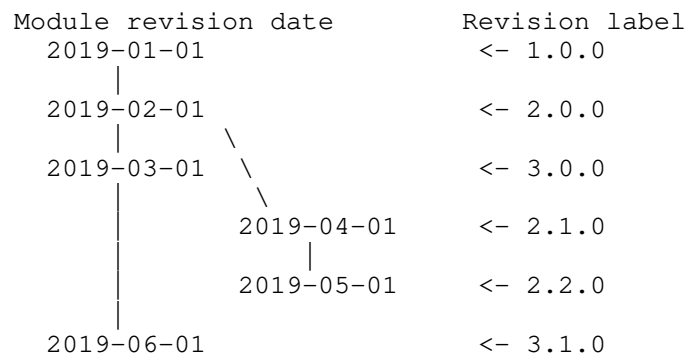
The ietf-yang-revision module specifies the YANG extension statement "status-description" that can be used as a substatement of the status statement. The argument to this extension statement can contain freeform text to help readers of the module understand why the node was deprecated or made obsolete, when it is anticipated that the node will no longer be available for use, and potentially reference other schema elements that can be used instead. An example is shown below.

```
leaf imperial-temperature {
  type int64;
  units "degrees Fahrenheit";
  status deprecated {
    rev:status-description
      "Imperial measurements are being phased out in favor
      of their metric equivalents. Use metric-temperature
      instead.";
  }
  description
    "Temperature in degrees Fahrenheit.";
}
```

3.5. Examples for updating the YANG module revision history

The following diagram, explanation, and module history illustrates how the branched revision history, "nbc-changes" extension statement, and "revision-label" extension statement could be used:

Example YANG module with branched revision history.



The tree diagram above illustrates how an example module's version history might evolve, over time. For example, the tree might represent the following changes, listed in chronological order from oldest revision to newest:

Example module, revision 2019-06-01:

```
module example-module {  
  
    namespace "name-space";  
    prefix "prefix-name";  
  
    import ietf-yang-revisions { prefix "rev"; }  
  
    description  
        "to be completed";  
  
    revision 2019-06-01 {  
        rev:revision-label "3.1.0";  
        rev:nbc-changes;  
        description "Add new functionality.";  
    }  
  
    revision 2019-04-01 {  
        rev:revision-label "3.0.0";  
        description  
            "Add new functionality. Remove some deprecated nodes.";  
    }  
  
    revision 2019-02-01 {  
        rev:revision-label "2.0.0";  
        rev:nbc-changes;  
        description "Apply bugfix to pattern statement";  
    }  
  
    revision 2019-01-01 {  
        rev:revision-label "1.0.0";  
        description "Initial revision";  
    }  
  
    //YANG module definition starts here
```

Example module, revision 2019-05-01:

```
module example-module {  
  
    namespace "name-space";  
    prefix "prefix-name";  
  
    import ietf-yang-revisions { prefix "semver"; }  
  
    description  
        "to be completed";  
  
    revision 2019-05-01 {  
        rev:revision-label "2.2.0";  
        description "Backwards-compatible bugfix to enhancement.";  
    }  
  
    revision 2019-03-01 {  
        rev:revision-label "2.1.0";  
        description "Apply enhancement to older release train.";  
    }  
  
    revision 2019-02-01 {  
        rev:revision-label "2.0.0";  
        rev:nbc-changes;  
        description "Apply bugfix to pattern statement";  
    }  
  
    revision 2019-01-01 {  
        rev:revision-label "1.0.0";  
        description "Initial revision";  
    }  
  
    //YANG module definition starts here
```

4. Import by derived revision

RFC 7950 allows YANG module "import" statements to optionally require the imported module to have a particular revision date. In practice, importing a module with an exact revision date is often too restrictive because it requires the importing module to be updated whenever any change to the imported module occurs. The alternative choice of using an import statement without any revision date statement is also not ideal because the importing module may not work with all possible revisions of the imported module.

Instead, it is desirable for a importing module to specify a "minimum required revision" of a module that it is compatible with, based on

the assumption that later revisions derived from that "minimum required revision" are also likely to be compatible. Many possible changes to a YANG module do not break importing modules, even if the changes themselves are not strictly backwards-compatible. E.g., fixing an incorrect pattern statement or description for a leaf would not break an import, changing the name of a leaf could break an import but frequently would not, but removing a container would break imports if that container is augmented by another module.

The ietf-revisions module defines the "revision-or-derived" extension statement, a substatement to the YANG "import" statement, to allow for a "minimum required revision" to be specified during import:

The argument to the "revision-or-derived" extension statement is a revision date or a revision label.

A particular revision of an imported module satisfies an import's "revision-or-derived" extension statement if the imported module's revision history contains a revision statement with a matching revision date or revision label.

An "import" statement MUST NOT contain both a "revision-or-derived" extension statement and a "revision-date" statement.

The "revision-or-derived" extension statement MAY be specified multiple times, allowing the import to use any module revision that satisfies at least one of the "revision-or-derived" extension statements.

The "revision-or-derived" extension statement does not guarantee that all module revisions that satisfy an import statement are necessarily compatible, it only gives an indication that the revisions are more likely to be compatible. Hence, NBC changes to an imported module may also require new revisions of any importing modules, updated to accommodate those changes, along with updated import "revision-or-derived" extension statements to depend on the updated imported module revision.

4.1. Module import examples

Consider the example module "example-module" from Section 3.5 that is hypothetically available in the following revision/label pairings: 2019-01-01/1.0.0, 2019-02-01/2.0.0, 2019-03-01/3.0.0, 2019-04-01/2.1.0, 2019-05-01/2.2.0 and 2019-06-01/3.1.0. The relationship between the revisions is as before:

Module revision date	Revision label
2019-01-01	<- 1.0.0
2019-02-01	<- 2.0.0
2019-03-01	<- 3.0.0
	2019-04-01 <- 2.1.0
	2019-05-01 <- 2.2.0
2019-06-01	<- 3.1.0

4.1.1. Example 1

This example selects module revisions that match, or are derived from the revision 2019-02-01. E.g., this dependency might be used if there was a new container added in revision 2019-02-01 that is augmented by the importing module. It includes revisions/labels: 2019-02-01/2.0.0, 2019-03-01/3.0.0, 2019-04-01/2.1.0, 2019-05-01/2.2.0 and 2019-06-01/3.1.0.

```
import example-module {
  ver:revision-or-derived 2019-02-01;
}
```

Alternatively, the first example could have used the revision label "1.0.0" instead, which selects the same set of revisions/versions.

```
import example-module {
  ver:revision-or-derived 1.0.0;
}
```

4.1.2. Example 2

This example selects module revisions that are derived from 2019-04-01 by using the revision label 2.1.0. It includes revisions/labels: 2019-04-01/2.1.0 and 2019-05-01/2.2.0. Even though 2019-06-01/3.1.0 has a higher revision label version number than 2019-04-01/2.1.0 it is not a derived revision, and hence it is not a valid revision for import.

```
import example-module {
  ver:revision-or-derived 2.1.0;
}
```

4.1.3. Example 3

This example selects revisions derived from either 2019-04-01 or 2019-06-01. It includes revisions/labels: 2019-04-01/2.1.0, 2019-05-01/2.2.0, and 2019-06-01/3.1.0.

```
import example-module {  
  ver:revision-or-derived 2019-04-01;  
  ver:revision-or-derived 2019-06-01;  
}
```

5. Updates to ietf-yang-library

YANG library [RFC7895] [RFC8525] is modified to support the new module update rules in three ways.

5.1. Advertising revision-label

The ietf-yang-revisions YANG module augments the "module" list in ietf-yang-library with a "revision-label" leaf to optionally declare the revision label associated with the particular revision of each module.

5.2. Resolving ambiguous module imports

A YANG datastore schema, defined in [RFC8525], can specify multiple revisions of a YANG module in the schema using the "import-only" list, with the requirement from [RFC7950] that only a single revision of a YANG module may be implemented.

If a YANG module import statement does not specify a specific revision within the datastore schema then it could be ambiguous as to which module revision the import statement should resolve to. Hence, a datastore schema constructed by a client using the information contained in YANG library may not exactly match the datastore schema actually used by the server.

The following two rules remove the ambiguity:

If a module import statement could resolve to more than one module revision defined in the datastore schema, and one of those revisions is implemented (i.e., not an "import-only" module), then the import statement MUST resolve to the revision of the module that is defined as being implemented by the datastore schema.

If a module import statement could resolve to more than one module revision defined in the datastore schema, and none of those revisions

are implemented, then the import MUST resolve to the module revision with the latest revision date.

5.3. Reporting how deprecated and obsolete nodes are handled

The ietf-yang-revisions YANG module augments YANG library with two leaves to allow a server to report how it handles status "deprecated" and status "obsolete" nodes. The leaves are:

deprecated-nodes-implemented: If present, this leaf indicates that all schema nodes with a status "deprecated" child statement are implemented equivalently as if they had status "current", or otherwise deviations MUST be used to explicitly remove "deprecated" nodes from the schema. If this leaf is absent then the behavior is unspecified.

obsolete-nodes-absent: If present, this leaf indicates that the server does not implement any status "obsolete" nodes. If this leaf is absent then the behaviour is unspecified.

Servers SHOULD set both the "deprecated-nodes-implemented" and "obsolete-nodes-absent" leaves.

If a server does not set the "deprecated-nodes-implemented" leaf, then clients MUST NOT rely solely on the "rev:nbc-changes" statements to determine whether two module revisions are backwards-compatible, and MUST also consider whether the status of any nodes has changed to "deprecated" and whether those nodes are implemented by the server.

6. Versioning of YANG instance data

Instance data sets [I-D.ietf-netmod-yang-instance-file-format] do not directly make use of the updated revision handling rules described in this document, as compatibility for instance data is undefined.

However, instance data specifies the content-schema of the data-set. This schema SHOULD make use of versioning using revision dates and/or revision labels for the individual YANG modules that comprise the schema or potentially for the entire schema itself (e.g., [I-D.rwilton-netmod-yang-packages]).

In this way, the versioning of a content-schema associated with an instance data set may help a client to determine whether the instance data could also be used in conjunction with other revisions of the YANG schema, or other revisions of the modules that define the schema.

7. Guidelines for using the YANG module update rules

The following text updates section 4.7 of [RFC8407] to revise the guidelines for updating YANG modules.

7.1. Guidelines for YANG module authors

NBC changes to YANG modules may cause problems to clients, who are consumers of YANG models, and hence YANG module authors are RECOMMENDED to minimize NBC changes and keep changes BC whenever possible.

When NBC changes are introduced, consideration should be given to the impact on clients and YANG module authors SHOULD try to mitigate that impact.

A "rev:nbc-changes" statement SHOULD be added only if there are NBC changes relative to the previous revision.

Removing old revision statements from a module's revision history could break import by revision, and hence it is RECOMMENDED to retain them. If all dependencies have been updated to not import specific revisions of a module, then the corresponding revision statements can be removed from that module. An alternative solution, if the revision section is too long, would be remove, or curtail, the older description statements associated with the previous revisions.

The "ver:revision-or-derived" extension should be used in YANG module imports to indicate revision dependencies between modules in preference to the "revision-date" statement, which causes overly strict import dependencies and SHOULD NOT be used.

A module that includes submodules MUST use the "revision-date" statement to include specific submodule revisions. Changing a module's include statements to include different submodule revisions requires a new revision of the module.

7.1.1. Making non-backwards-compatible changes to a YANG module

There are various valid situations where a YANG module has to be modified in an NBC way. Here are the different ways in which this can be done:

- o NBC changes can be sometimes be done incrementally using the "deprecated" status to provide clients time to adapt to NBC changes.

- o NBC changes are done at once, i.e. without using "status" statements. Depending on the change, this may have a big impact on clients.
- o If the server can support multiple versions of the YANG module or of YANG packages(as specified in [I-D.rwilton-netmod-yang-packages]), and allows the client to select the version (as per [I-D.wilton-netmod-yang-ver-selection]), then NBC changes MAY be done without using "status" statements. Clients would be required to select the version which they support and the NBC change would have no impact on them

Here are some guidelines on how non-backwards-compatible changes can be made incrementally, with the assumption that deprecated nodes are implemented by the server, and obsolete nodes are not:

1. The changes should be made gradually, e.g. a data node's status SHOULD NOT be changed directly from "current" to "obsolete" (see Section 4.7 of [RFC8407]), instead the status SHOULD first be marked "deprecated" and then when support is removed its status MUST be changed to "obsolete". Instead of using the "obsolete" status, the data node MAY be removed from the model but this has the risk of breaking modules which import the modified module.
2. The new "status-description" extension statement SHOULD be used for nodes which are "obsolete" or "deprecated".
3. For status "deprecated", the "status-description" SHOULD also indicate until when support for the node is guaranteed (if known). If there is a replacement data node, rpc, action or notification for the deprecated node, this SHOULD be stated in the "status-description". The reason for deprecating the node can also be included in the "status-description" if it is deemed to be of potential interest to the user.
4. For status "obsolete", it is RECOMMENDED to keep the "status-description" information, from when the node had status "deprecated, which is still relevant.
5. When obsoleting or deprecating data nodes, the "deprecated" or "obsolete" status SHOULD be applied at the highest possible level in the data tree with an appropriate "status-description" statement. For clarity, the "status" statement SHOULD also be applied to all descendent data nodes, but the "status-description" statement does not need to be repeated if it does not introduce any additional information.

See Appendix A.1 for examples on how NBC changes can be made.

7.2. Versioning Considerations for Clients

Guidelines for clients of modules using the new module revision update procedure:

- o Clients SHOULD be liberal when processing data received from a server. For example, the server may have increased the range of an operational node causing the client to receive a value which is outside the range of the YANG model revision it was coded against.
- o Clients SHOULD monitor changes to published YANG modules through their revision history, and use appropriate tooling to understand the specific changes between module revision. In particular, clients SHOULD NOT migrate to NBC revisions of a module without understanding any potential impact of the specific NBC changes.
- o Clients SHOULD plan to make changes to match published status changes. When a node's status changes from "current" to "deprecated", clients SHOULD plan to stop using that node in a timely fashion. When a node's status changes to "obsolete", clients MUST stop using that node.

8. Module Versioning Extension YANG Modules

YANG module with extension statements for annotating NBC changes, revision label, status description, and importing by version.

```
<CODE BEGINS> file "ietf-yang-revisions@2019-05-02.yang"
module ietf-yang-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-revisions";
  prefix rev;

  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netmod/>
    WG List:  <mailto:netmod@ietf.org>

    Author:   Benoit Claise
              <mailto:bclaise@cisco.com>

    Author:   Joe Clarke
              <mailto:jclarke@cisco.com>

    Author:   Reshad Rahman
```

```
<mailto:rrahman@cisco.com>

Author:   Robert Wilton
         <mailto:rwilton@cisco.com>

Author:   Kevin D'Souza
         <mailto:kd6913@att.com>

Author:   Balazs Lengyel
         <mailto:balazs.lengyel@ericsson.com>

Author:   Jason Sterne
         <mailto:jason.sterne@nokia.com>";
description
  "This YANG 1.1 module contains definitions and extensions to
  support updated YANG revision handling.";

revision 2019-05-02 {
  description
    "Initial version.  Derived from ietf-semver.yang@2019-02-17.";
  reference
    "draft-verdt-netmod-module-versioning: Updated YANG Module
    Revision Handling";
}

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a specific date in YYYY-MM-DD format.
    TODO - Import and reuse type from 6991-bis";
}

typedef label-string {
  type string {
    length "1..255";
    pattern '^[^s@]+';
    pattern '\d{4}-\d{2}-\d{2}' {
      modifier invert-match;
    }
  }
  description
    "A label associated with a YANG revision.

    Excludes spaces and '@'.  Cannot match revision-date.";
  reference
    "draft-verdt-netmod-yang-module-versioning: Revision label";
```

```
}

typedef revision-date-or-label {
  type union {
    type revision-identifier;
    type label-string;
  }
  description
    "Represents either a YANG revision date or a revision label";
}

extension nbc-changes {
  description
    "This statement is used to indicate YANG module revisions that
    contain non-backwards-compatible changes.

    Each 'revision' statement MAY have a single 'nbc-changes'
    substatement.

    If a revision of a YANG module contains changes, relative to
    the preceding revision in the revision history, that do not
    conform to the module update rules defined in RFC-XXX, then
    the 'nbc-changes' statement MUST be added as a substatement to
    the revision statement.

    Conversely, if a revision of a YANG module only contains
    changes, relative to the preceding revision in the revision
    history, that are classified as 'backwards-compatible' then
    the revision statement MUST NOT contain any 'nbc-changes'
    substatement.";
  reference
    "draft-verdt-netmod-module-versioning: nbc-changes revision
    extension statement";
}

extension revision-label {
  argument label-string;
  description
    "The revision label can be used to provide an additional
    versioning identifier associated with the revision. E.g., one
    option for a versioning scheme that could be used is [TODO -
    Reference semver draft].

    Each 'revision' statement MAY have a single 'revision-label'
    substatement.

    Revision labels MUST be unique amongst all revisions of a
    module.";
```

```
reference
  "draft-verdt-netmod-module-versioning: Revision label";
}

extension revision-or-derived {
  argument revision-date-or-label;
  description
    "Restricts the revision of the module that may be imported to
    one that matches or is derived from the specified
    revision-date or revision-nlabel.

    The argument value MUST conform to the
    'revision-date-or-label' defined type.

    Each 'import' statement MAY have one or more
    'revision-or-derived' substatements.  If specified multiple
    times, then any module revision that satisfies at least one of
    the 'revision-or-derived' statements is an acceptable revision
    for import.

    An 'import' statement MUST NOT contain both a
    'revision-or-derived' extension statement and a
    'revision-date' statement.

    A particular revision of an imported module satisfies an
    import's 'revision-or-derived' extension statement if the
    imported module's revision history contains a revision
    statement with a matching revision date or revision label.

    The 'revision-or-derived' extension statement does not
    guarantee that all module revisions that satisfy an import
    statement are necessarily compatible, it only gives an
    indication that the revisions are more likely to be
    compatible.";

  reference
    "draft-verdt-netmod-yang-module-versioning: Import by derived
    revision";
}

extension status-description {
  argument description;

  description
    "Freeform text that describes why a given node has been
    deprecated or made obsolete.  E.g., the description could be
    used to give the reason for removal, or it could point to an
    alternative schema elements that can be used in lieu of the
```

given node.

Each 'status' statement MAY have a single 'status-description' substatement.";

```
reference
  "draft-verdt-netmod-yang-module-versioning: YANG status
  description extension";
}
}
<CODE ENDS>
```

YANG module with augmentations to YANG Library to revision labels

```
<CODE BEGINS> file "ietf-yl-revisions@2019-05-02.yang"
module ietf-yl-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yl-revisions";
  prefix yl-rev;

  import ietf-revisions {
    prefix rev;
  }

  import ietf-yang-library {
    prefix yanglib;
  }

  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netmod/>
    WG List:  <mailto:netmod@ietf.org>

    Author:   Benoit Claise
              <mailto:bclaise@cisco.com>

    Author:   Joe Clarke
              <mailto:jclarke@cisco.com>

    Author:   Reshad Rahman
              <mailto:rrahman@cisco.com>

    Author:   Robert Wilton
              <mailto:rwilton@cisco.com>

    Author:   Kevin D'Souza
              <mailto:kd6913@att.com>
```



```

    Author:   Balazs Lengyel
              <mailto:balazs.lengyel@ericsson.com>

    Author:   Jason Sterne
              <mailto:jason.sterne@nokia.com>";
description
  "This module contains augmentations to YANG Library to add module
  level revision label and to provide an indication of how
  deprecated and obsolete nodes are handled by the server.";

revision 2019-05-02 {
  description
    "Initial revision, derived from ietf-yl-semver~2019-02-17";
  reference
    "draft-verdt-netmod-module-versioning: Updated YANG Module
    Revision Handling";
}

augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
  description
    "Augmentation modules with a revision label";
  leaf revision-label {
    type rev:label-string;
    description
      "The revision label associated with this module revision.
      The label MUST match the rev:label value in the specific
      revision of the module loaded in this module-set.";
    reference
      "draft-verdt-netmod-module-versioning: Updated YANG Module
      Revision Handling";
  }
}

augment "/yanglib:yang-library/yanglib:schema" {
  description
    "Augmentations to the ietf-yang-library module to indicate how
    deprecated and obsoleted nodes are handled for each datastore
    schema supported by the server.";

  leaf deprecated-nodes-implemented {
    type empty;
    description
      "If present, this leaf indicates that all schema nodes with a
      status 'deprecated' child statement are implemented
      equivalently as if they had status 'current', or otherwise
      deviations MUST be used to explicitly remove 'deprecated'
      nodes from the schema. If this leaf is absent then the
      behavior is unspecified.";
  }
}
```

```
        reference
          "draft-verdt-netmod-yang-semver: Reporting how deprecated and
          obsolete nodes are handled";
      }
    leaf obsolete-nodes-absent {
      type empty;
      description
        "If present, this leaf indicates that the server does not
        implement any status 'obsolete' nodes. If this leaf is
        absent then the behaviour is unspecified.";
      reference
        "draft-verdt-netmod-yang-semver: Reporting how deprecated and
        obsolete nodes are handled";
    }
  }
}
<CODE ENDS>
```

9. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following individuals are (or have been) members of the design team and have worked on the YANG versioning project:

- o Balazs Lengyel
- o Benoit Claise
- o Ebben Aries
- o Jason Sterne
- o Joe Clarke
- o Juergen Schoenwaelder
- o Mahesh Jethanandani
- o Michael (Wangzitao)
- o Qin Wu
- o Reshad Rahman
- o Rob Wilton

The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update].

Discussions on the use of Semver for YANG versioning has been held with authors of the OpenConfig YANG models. We would like thank both Anees Shaikh and Rob Shakir for their input into this problem space.

10. Security Considerations

The document does not define any new protocol or data model. There are no security impacts.

11. IANA Considerations

11.1. YANG Module Registrations

The following YANG module is requested to be registered in the "IANA Module Names" registry:

The ietf-yang-revisions module:

Name: ietf-yang-revisions

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-revisions

Prefix: rev

Reference: [RFCXXXX]

The ietf-yl-revisions module:

Name: ietf-yl-revisions

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yl-revisions

Prefix: yl-rev

Reference: [RFCXXXX]

12. References

12.1. Normative References

[I-D.verdt-netmod-yang-versioning-reqs]

Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-02 (work in progress), November 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>>.
- [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>>.
- [RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <<https://www.rfc-editor.org/info/rfc8407>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.

12.2. Informative References

- [I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", draft-clacla-netmod-yang-model-update-06 (work in progress), July 2018.
- [I-D.ietf-netmod-yang-instance-file-format]
Lengyel, B. and B. Claise, "YANG Instance Data File Format", draft-ietf-netmod-yang-instance-file-format-02 (work in progress), February 2019.
- [I-D.rwilton-netmod-yang-packages]
Wilton, R., "YANG Packages", draft-rwilton-netmod-yang-packages-01 (work in progress), March 2019.
- [I-D.verdt-netmod-yang-solutions]
Wilton, R., "YANG Versioning Potential Solutions", draft-verdt-netmod-yang-solutions-00 (work in progress), October 2018.

[I-D.wilton-netmod-yang-ver-selection]

Wilton, R. and R. Rahman, "YANG Schema Version Selection",
draft-wilton-netmod-yang-ver-selection-00 (work in
progress), March 2019.

Appendix A. Appendix

A.1. Examples of guidelines for making NBC changes to a YANG module

Examples of NBC changes include:

- o Deleting a data node, or changing it to status obsolete.
- o Changing the name, type, or units of a data node.
- o Modifying the description in a way that changes the semantic meaning of the data node.
- o Any changes that change or reduce the allowed value set of the data node, either through changes in the type definition, or the addition or changes to "must" statements, or changes in the description.
- o Adding or modifying "when" statements that reduce when the data node is available in the schema.
- o Making the statement conditional on if-feature.

The following sections give guidance for how some of these NBC changes could be made to a YANG module:

A.1.1. Removing a data node

Removing a leaf or container from the data tree, e.g. because support for the corresponding feature is being removed:

1. The node's status is changed to "deprecated" and it is supported for at least one year. This is a BC change.
2. When the node is not available anymore, its status is changed to "obsolete" and the "status-description" updated, this is an NBC change. The "status-description" is used to explain why the node is not available anymore.

If the server can support NBC versions of the YANG module simultaneously using version selection, then the changes can be done immediately:

1. The new revision of the YANG module has the node's status changed to "obsolete" and the "status-description" updated, this is an NBC change.
2. Clients which require the data node select the older module revision

A.1.2. Changing the type of a leaf node

Changing the type of a leaf-node. e.g. consider a "vpn-id" node of type integer being changed to a string:

1. The status of node "vpn-id" is changed to "deprecated" and the node should be available for at least one year. This is a BC change.
2. A new node, e.g. "vpn-name", of type string is added to the same location as the existing node "vpn-id". This new node has status "current" and its description explains that it is replacing node "vpn-id".
3. During the period of time where both nodes are available, how the server behaves when either node is set is outside the scope of this document and will vary on a case by case basis. Here are some options:
 1. A server may prevent the new node from being set if the old node is already set (and vice-versa). The new node may have a when statement to achieve this. The old node must not have a when statement since this would be an NBC change, but the server could reject the old node from being set if the new node is already set.
 2. If the new node is set and a client does a get or get-config operation on the old node, the server could map the value. For example, if the new node "vpn-name" has value "123" then the server could return integer value 123 for the old node "vpn-id". However, if the value can not be mapped, we need a way of returning "unsupported" TBD.
4. When node "vpn-id" is not available anymore, its status is changed to "obsolete" and the "status-description" is updated. This is an NBC change.

If the server can support NBC versions of the YANG module simultaneously using version selection, then the changes can be done immediately:

1. In the new revision of the YANG module, the status of node "vpn-id" is changed to "obsolete". This is an NBC change.
2. New node "vpn-name" is added to the same location as described above.
3. Clients which require the data node select the older module revision
4. A server should not map between the nodes "vpn-id" and "vpn-name", i.e. if a client creates a data instance with "vpn-name" then that data instance should not be visible to a client using a module revision which has "vpn-id" (and vice-versa).

A.1.3. Reducing the range of a leaf node

Reducing the range of values of a leaf-node. e.g. consider a "vpn-id" node of type integer being changed from type uint32 to type uint16:

1. If all values which are being removed were never supported, e.g. if a vpn-id of 65536 or higher was never accepted, this is a BC change for the functionality (no functionality change). Even if it is an NBC change for the YANG model, there should be no impact for clients using that YANG model.
2. If one or more values being removed was previously supported, e.g. if a vpn-id of 65536 was accepted previously, this is an NBC change for the YANG model. Clients using the old YANG model will be impacted, so a change of this nature should be done carefully, e.g. by using the steps described in Appendix A.1.2

A.1.4. Changing the key of a list

Changing the key of a list has a big impact to the client. For example, consider a "sessions" list which has a key "interface" and there is a need to change the key to "dest-address", such a change can be done in steps:

1. The status of list "sessions" is changed to "deprecated" and the list should be available for at least one year. This is a BC change.
2. A new list is created in the same location with the same data but with "dest-address" as key. Finding an appropriate name for the new list can be tricky especially if the name of the existing list was perfect. In this case the new list is called "sessions-address", has status "current" and its description should explain that it is replacing list "session".

3. During the period of time where both lists are available, how the server behaves when either list is set is outside the scope of this document and will vary on a case by case basis. Here are some options:
 1. A server could prevent the new list from being set if the old list already has entries (and vice-versa).
 2. If the new list is set and a client does a get or get-config operation on the old list, the server could map the entries. However if the new list has entries which would lead to duplicate keys in the old list, the mapping can not be done.
4. When list "sessions" is not available anymore, its status is changed to "obsolete" and the "status-description" is updated. This is an NBC change.

If the server can support NBC versions of the YANG module simultaneously using version selection, then the changes can be done immediately:

1. The new revision of the YANG module has the list "sessions" modified to have "dest-address" as key, this is an NBC change.
2. Clients which require the previous functionality select the older module revision

A.1.5. Renaming a node

A leaf-node or a container may be renamed, either due to a spelling error in the previous name or because of a better name. For example a node "ip-adress" could be renamed to "ip-address":

1. The status of the existing node "ip-adress" is changed to "deprecated" and the node should be available for at least one year. This is a BC change.
2. The new node "ip-address" is added to the same location as the existing node "ip-adress". This new node has status "current" and its description should explain that it is replacing node "ip-adress".
3. During the period of time where both nodes are available, how the server behaves when either node is set is outside the scope of this document and will vary on a case by case basis. Here are some options:

1. A server could prevent the new node from being set if the old node is already set (and vice-versa). The new node could have a when statement to achieve this. The old node must not have a when statement since this would be an NBC change, but the server could reject the old node from being set if the new node is already set.
2. If the new node is set and a client does a get or get-config operation on the old node, the server could use the value of the new node. For example, if the new node "ip-address" has value X then the server may return value X for the old node "ip-adress".
4. When node "ip-adress" is not available anymore, its status is changed to "obsolete" and the "status-description" is updated. This is an NBC change.

If the server can support NBC versions of the YANG module simultaneously using version selection, then the changes can be done immediately:

1. The new revision of the YANG module has the node with the new name replacing the node with the old name, this is an NBC change.
2. Clients which require the previous node name select the older module revision

A.1.6. Changing a default value

Authors' Addresses

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Reshad Rahman
Cisco Systems, Inc.

Email: rrahman@cisco.com

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Balazs Lengyel
Ericsson
Magyar Tudosok Korutja
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
United States of America

Email: kd6913@att.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2020

R. Wilton, Ed.
Cisco Systems, Inc.
July 3, 2019

YANG Versioning Solution Overview
draft-verdt-netmod-yang-solutions-01

Abstract

This temporary document gives a brief overview of the different drafts that comprise a full solution to the YANG versioning requirements draft. The purpose of this draft is to help readers understand how the discrete parts of the YANG versioning solution fit together during working group development of the solution drafts.

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 4, 2020.

Copyright Notice

Copyright (c) 2019 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. Summary of requirements	2
3. Solution Drafts	4
3.1. Updated YANG Module Revision Handling	4
3.2. Module semantic version number scheme	5
3.3. Versioned YANG packages	5
3.4. Protocol operations for package version selection	6
3.5. YANG schema comparison tooling	6
4. Contributors	7
5. Security Considerations	7
6. IANA Considerations	8
7. References	8
7.1. Normative References	8
7.2. Informative References	8
Author's Address	9

1. Introduction

[I-D.ietf-netmod-yang-versioning-reqs] documents the requirements for any solution to the YANG versioning problem. Chapter 5 lists the formal requirements that a complete solution requires.

The aim of this draft is to help readers understand how the different solution drafts fit together, and also which drafts contribute solutions to particular individual requirements. The overall solution comprises five individual drafts:

1. [I-D.verdt-netmod-yang-module-versioning]
2. Module semantic version number scheme (not yet published)
3. [I-D.rwilton-netmod-yang-packages]
4. [I-D.wilton-netmod-yang-ver-selection]
5. YANG schema comparison tooling (not yet published)

Open issues, across all of the solution drafts are tracked at <https://github.com/netmod-wg/yang-ver-dt/issues>.

2. Summary of requirements

The requirements are formally documented in section 5 of [I-D.ietf-netmod-yang-versioning-reqs]. A shortened, non normative, summary of each of the requirements (using the same requirement

numbers) is provided below to help understand how the solutions drafts address the particular requirements.

Req 1.1 - MUST allow nbc updates to a module without breaking imports.

Req 1.2 - MUST allow nbc updates to a module without affecting existing client code using only unchanged data nodes.

Req 1.3 - MUST support import statement restricted to only some revisions.

Req 1.4 - MUST allow for fixes to non-latest published modules.

Req 2.1 - MUST be able to determine if two arbitrary versions of any module are backwards-compatible.

Req 2.2 - SHOULD be able to determine if two arbitrary versions of any data node are backwards-compatible.

Req 3.1 - MUST allow servers to support existing clients.

Req 3.2 - MUST allow simultaneously support of clients using different (perhaps restricted) revisions.

Req 3.3 - MAY assume clients can handle unexpected instance data gracefully.

Req 4.1 - MUST provide a way to indicate if deprecated nodes are implemented.

Req 4.2 - MUST be able to document the reason for data node lifecycle changes, and possible alternative data nodes.

Req 4.3 - MUST be able to forewarn of future data node lifecycle changes.

Req 5.1 - MUST provide guidance on how to use the new scheme.

Req 5.2 - MUST provide, and document, an upgrade path from existing YANG/protocols.

Req 5.3 - MUST consider the impact of versioning on instance data.

3. Solution Drafts

The complete solution to the YANG versioning requirements comprises five solution drafts, that are summarized below.

3.1. Updated YANG Module Revision Handling

In summary, [I-D.verdt-netmod-yang-module-versioning] specifies minimal extensions and updates to the YANG language, YANG Library, and YANG author guidelines to provide more flexible YANG module revision handling. The intent is that these changes and extensions could be folded into future revisions of the updated specifications. The draft provides a solution for all requirements except Req 2.2, Req 3.1 and Req 3.2.

The extensions and changes in the draft can be summarized thus:

- o It defines a YANG extension statement to indicate where non-backwards-compatible changes have occurred in a module's revision history.
- o It relaxes the rules for the module revision history to allow for a non-linear module revision history. I.e., any given module revision may have multiple revisions directly derived from it.
- o It defines a new import extension statement that restricts the allowed module revisions that satisfy the import to only those derived from a specified module revision.
- o It defines a revision label extension statement to allow an informative name to be associated with a particular revision date, and to be used in import statements, YANG module filenames, and is available in YANG library. One example of how the revision label could be used is to associate a semantic versioning scheme to YANG module revisions.
- o It updates the YANG rules for changes between module revisions that are allowed to be classified as backwards-compatible. In particular, marking a node as obsolete is no longer classified as a backwards compatible change.
- o It provides updated guidance on how servers handle deprecated and obsolete YANG nodes and augments YANG library with additional leaves to report the server's behavior to clients.
- o It provides an extension statement to allow a description statement to be associated with a YANG status statement, providing more information about why the status has changed.

- o It defines how versioning relates to YANG instance data.
- o It refines the guidelines for updating modules, taking into consideration that non-backwards-compatible changes are sometimes necessary for various pragmatic reasons.

3.2. Module semantic version number scheme

[TODO - Insert draft ref] defines a semantic versioning scheme derived from the semver.org 2.0.0 specification that can be used in conjunction with the revision label extension statement defined in Section 3.1 to allow semantic version numbers to be used to manage the revision lifecycle of YANG modules. This draft provides an enhanced solution for Req 2.1, but organizations authoring modules are not obliged to use this specific versioning scheme, and could choose a different overlaid versioning scheme, or none at all and rely solely on revision dates.

The aims of the YANG semantic versioning scheme are:

To generally allow clients to determine whether NBC changes have occurred between two revisions from the version number alone, without having to check the full revision history.

To give a more informative identifier for a branched revision history over revision dates alone.

To allow revision branches that contain fixes for published non-latest releases.

3.3. Versioned YANG packages

The two previous drafts address version and revision management of individual modules. [I-D.rwilton-netmod-yang-packages] provides a mechanism to group a set of related YANG modules revisions together, into a construct called a YANG package, and to apply a version scheme to the group.

The core part of this draft are YANG module definitions that define a YANG package, that are used as an augmentation to YANG library, and also in YANG instance data documents for offline access.

The principle aims of the packages draft are:

To provide an alternative simpler mechanism to manage conformance of modules. Rather than checking conformance against a set of individual YANG module revisions, it should be easier to check for conformance against a much smaller set of YANG package versions.

To provide an easier mechanism for clients to check conformance with a server. Rather than downloading and comparing all individual module revisions, the client can just check whether the package version is compatible. The package definition could be retrieved and cached from multiple sources.

The YANG packages draft does not address any of the versioning requirements directly, but provides the foundation building blocks for the version selection solution, described in Section 3.4, that addresses Reqs 3.1 and 3.2.

This draft requires updates to accommodate the split between revision management in [I-D.verdt-netmod-yang-module-versioning] and the semantic versioning scheme.

3.4. Protocol operations for package version selection

[I-D.wilton-netmod-yang-ver-selection] specifies a solution for requirements 3.1 and 3.2 is via the use of [I-D.rwilton-netmod-yang-packages] and a protocol based version selection scheme that can be used by clients to choose a particular YANG datastore schema from the set of datastore schema that are supported by the server.

The version selection optionally allows:

Servers to support a single, selectable YANG package at a particular version, that is used for all NETCONF/RESTCONF interactions.

Servers to support multiple selectable YANG packages and package versions, with different clients able to concurrently access different packages and different package versions.

3.5. YANG schema comparison tooling

A tooling based solution is proposed for Req 2.2, that allows two YANG schema versions to be algorithmically compared, with the algorithm reporting the list of differences between the two YANG schema and whether each change is regarded as being backwards-compatible, or non-backwards-compatible. Annotations to the YANG modules, via the use of extension statements, may help improve the accuracy of the comparison algorithm, particularly for statements that are very hard to algorithmically classify the scope of any differences (e.g., a change in the semantic behaviour of a data node defined via modifications to the associated YANG description statement). Given that Req 2.2 is a softer requirement, and

practical experience with the tooling is required, it is proposed that this work is deferred at this time.

When comparing a module schema, a tool would also be able to take into account enabled features, deviations, and the subset of the schema being used by the client. This would allow a tooling based approach to give a more accurate answer as to whether a client would be affected when upgrading between two software versions, than looking at the revision history, or comparing semantic version numbers.

4. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following individuals are (or have been) members of that design team and have contributed to defining the problem and specifying the requirements:

- o Balazs Lengyel
- o Benoit Claise
- o Ebben Aries
- o Jason Sterne
- o Joe Clarke
- o Juergen Schoenwaelder
- o Mahesh Jethanandani
- o Michael (Wangzitao)
- o Qin Wu
- o Reshad Rahman
- o Rob Wilton
- o Susan Hares

5. Security Considerations

The document does not define any new protocol or data model. There is no security impact.

6. IANA Considerations

None

7. References

7.1. Normative References

- [I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-ietf-netmod-yang-versioning-reqs-00 (work in progress), March 2019.
- [I-D.rwilton-netmod-yang-packages]
Wilton, R., "YANG Packages", draft-rwilton-netmod-yang-packages-01 (work in progress), March 2019.
- [I-D.verdt-netmod-yang-module-versioning]
Claise, B., Clarke, J., Rahman, R., Wilton, R., Lengyel, B., Sterne, J., and K. D'Souza, "Updated YANG Module Revision Handling", draft-verdt-netmod-yang-module-versioning-00 (work in progress), July 2019.
- [I-D.wilton-netmod-yang-ver-selection]
Wilton, R. and R. Rahman, "YANG Schema Version Selection", draft-wilton-netmod-yang-ver-selection-00 (work in progress), March 2019.
- [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>>.

7.2. Informative References

- [RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <<https://www.rfc-editor.org/info/rfc8049>>.
- [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>>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki,
"YANG Data Model for L3VPN Service Delivery", RFC 8299,
DOI 10.17487/RFC8299, January 2018,
<<https://www.rfc-editor.org/info/rfc8299>>.

Author's Address

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

R. Wilton
R. Rahman
Cisco Systems, Inc.
March 11, 2019

YANG Schema Version Selection
draft-wilton-netmod-yang-ver-selection-00

Abstract

This document defines protocol mechanisms to allow clients to choose which YANG schema to use for interactions with a server, out of the available YANG schema supported by a server. The provided functionality allow servers to support clients in a backwards compatible way, at the same time allowing for non-backwards-compatible updates to YANG modules.

This draft provides a solution to YANG versioning requirements 3.1 and 3.2.

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 September 12, 2019.

Copyright Notice

Copyright (c) 2019 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. Terminology and Conventions	2
2. Introduction	3
3. Background	4
4. Objectives	4
5. Solution Overview	5
6. Version selection from a server perspective	6
7. Version selection from a clients perspective	7
8. Limitations of the solution	7
9. Schema Version Selection YANG module	8
10. YANG Module	9
11. Security Considerations	13
12. IANA Considerations	14
13. Open Questions/Issues	14
14. Acknowledgements	14
15. References	14
15.1. Normative References	14
15.2. Informative References	15
Authors' Addresses	16

1. Terminology and Conventions

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.

This document uses terminology introduced in the YANG versioning requirements draft [I-D.verdt-netmod-yang-versioning-reqs].

This document also makes of the following terminology introduced in the Network Management Datastore Architecture [RFC8342]:

- o datastore schema

In addition, this document makes use of the following terminology:

- o bc: Used as an abbreviation for a backwards-compatible change.
- o nbc: Used as an abbreviation for a non-backwards-compatible change.

- o editorial change: A backwards-compatible change that does not change the YANG module semantics in any way.
- o YANG schema: The combined set of schema nodes for a set of YANG module revisions, taking into consideration any deviations and enabled features.
- o versioned schema: A YANG schema with an associated YANG semantic version number, e.g., as might be described by a YANG package.
- o schema set: A set of related versioned YANG schema, one for each datastore that is supported.

TODO - the bc/nbc/editorial terminology should probably be defined and referenced from the YANG module versioning solution draft. 'schema' and 'versioned schema' could be defined in the packages draft.

2. Introduction

This document describes how NETCONF and RESTCONF clients can choose a particular YANG schema they wish to choose to interact with a server with.

[I-D.verdt-netmod-yang-versioning-reqs] defines requirements that any solution to YANG versioning must have.

[I-D.verdt-netmod-yang-semver] specifies a partial solution to the YANG versioning requirements that focuses on using semantic versioning within individual YANG modules, but does not address all the requirements listed in the requirements draft. Of particular relevance here, requirements 3.1 and 3.2 are not addressed.

[I-D.rwilton-netmod-yang-packages] describes how sets of related YANG modules can be grouped together into a logical entity that is versioned using the YANG semantic versioning number scheme. Different packages can be defined for different sets of YANG modules, e.g., packages could be defined for the IETF YANG modules, OpenConfig YANG modules, a vendor's YANG modules. Different versions of these package definitions can be defined as the contents of these packages evolve over time, and as the versions of the YANG modules included in the package evolve.

This draft defines how YANG packages can be used to represent versioned datastore schema, and how clients can choose which versioned schemas to use during interactions with a device.

3. Background

There are three ways that the lifecycle of a data model can be managed:

1. Disallow all non-backwards-compatible updates to a YANG module. Broadly this is the approach adopted by [RFC7950], but it has been shown to be too inflexible in some cases. E.g. it makes it hard to fix bugs in a clean fashion - it is not clear that allowing two independent data nodes (one deprecated, one current) to configure the same underlying property is robustly backwards compatible in all scenarios, particularly if the value space and/or default values differ between the module revisions.
2. Allow non-backwards-compatible updates to YANG modules, and use a mechanism such as semantic version numbers to communicate the likely impact of any changes to module users, but require that clients handle non-backwards-compatible changes in servers by migrating to new versions of the modules. Without version selection, this is what the [I-D.verdt-netmod-yang-semver] approach likely achieves.
3. Allow non-backwards-compatible updates to YANG modules, but also provide mechanisms to allow servers to support multiple versions of YANG modules, and provide clients with some ability to select which versions of YANG modules they wish to interact with, subject to some reasonable constraints. This is the approach that this draft aims to address. It is worth noting that the idea of supporting multiple versions of an API is not new in the wider software industry, and there are many examples of where this approach has been successfully used.

4. Objectives

The goals of the schema version selection draft are:

- o To provide a mechanism where non-backwards-compatible changes and bug fixes can be made to YANG modules without forcing clients to immediately migrate to new versions of those modules as they get implemented.
- o To allow servers to support multiple versions of a particular YANG schema, and to allow clients to choose which YANG schema version to use when interoperating with the server. The aim here is to give operators more flexibility as to when they update their software.

- o To provide a mechanism to allow different YANG schema families (e.g., SDO models, OpenConfig models, Vendor models) to be supported by a server, and to allow clients to choose which YANG schema family is used to interoperate with the server.

The following points are non objective of this draft:

- o This draft does not provide a mechanism to allow clients to choose arbitrary sets of YANG module versions to interoperate with the server.
- o Servers are not required to concurrently support clients using different YANG schema families or versioned schema. A server MAY choose to only allow a single schema family or single versioned schema to be used by all clients.
- o There is no requirement for a server to support every published version of a YANG package, particularly if some package versions are backwards compatible. Clients are required to interoperate with backwards compatible updates of YANG modules. E.g., if a particular package was available in versions 1.0.0, 1.1.0, 1.2.0, 2.0.0, 3.0.0 and 3.1.0, then a server may choose to only support versions 1.2.0, 2.0.0, and 3.1.0, with the knowledge that all clients should be able to interoperate with the server.
- o There is no requirement to support all parts of all versioned schemas. For some nbc changes in modules, it is not possible for a server to support both the old and new module versions, and to convert between the two. Where appropriate deviations can be used, and otherwise an out of band mechanism is used to indicate where a mapping has failed.

5. Solution Overview

An overview the solution is as follows:

1. YANG packages are defined for the different versioned schema supported by a server:
 - * Separate packages can be defined for different families of schema, e.g., SDO, OpenConfig, or vendor native.
 - * Separate packages can be defined for each versioned schema within a schema family.
 - * Separate packages may be defined for different datastores, if the datastores use different datastore schema. For example, a

different datastore schema, and hence package, might be used for <operational> vs the conventional datastores.

2. Each server advertises, via an operational data model:
 - * All of the YANG packages that may be used during version selection. The packages can also be made available for offline consumption via instance data documents, as described in [I-D.rwilton-netmod-yang-packages].
 - * Grouped sets of versioned schema, where each set defines the versioned schema used by each supported datastore, and each versioned schema is represented by a YANG package instance.
3. Each server supports configuration to:
 - * Allow a client to configure which schema version set to use for the default NETCONF/RESTCONF connections.
 - * Allow a client to configure additional separate NETCONF and RESTCONF protocol instances, which use different schema version sets on those protocol instances.
 - * An RPC mechanism could also be defined to select schema, but is not currently discussed in this draft.
4. The server internally maps requests between the different protocol instances to the internal device implementation.

6. Version selection from a server perspective

The general premise of this solution is that servers generally implement one native schema, and the version selection scheme is used to support older version of that native schema and also foreign schema specified by external entities.

Overall the solution relies on the ability to map instance data between different schema versions. Depending on the scope of difference between the schema versions then some of these mappings may be very hard, or even impossible, to implement. Hence, there is still a strong incentive to try and minimize nbc changes between schema versions to minimize the mapping complexity.

Server implementations MUST serialize configuration requests across the different schema. The expectation is that this would be achieved by mapping all requests to the devices native schema version.

Datastore validation needs to be performed in two places, firstly in whichever schema a clients is interacting in, and secondly in the native schema for the device. This could have a negative performance impact.

Depending on the complexity of the mappings between schema versions, it may be necessary for the mappings to be stateful.

TODO - Figure out how hot fixes that slightly modify the schema are handled.

7. Version selection from a clients perspective

Clients can use configuration to choose which schema sets are available.

Clients cannot choose arbitrary individual YANG module versions, and are instead constrained by the versions that the server makes available.

Each client protocol connection is to one particular schema set. From that client session perspective it appears as if the client is interacting with a regular server. If the client queries YANG library that the version of YANG Library that is returned matches the schema set that is being used for that server instance.

The server may not support a schema with the exact version desired by the client, and they have to accept a later version that is backwards compatible with their desired version. Clients may also have to accept later schema versions that contain NBC fixes, although the assumption is that such nbc fixes should be designed to minimize the impact on clients.

There is no guarantee that servers will always be able to support all older schema versions. Deviations should be used where necessary to indicate that the server is unable to faithfully implement the older schema version.

If clients interact with a server using multiple versions, they should not expect that all data nodes in later module versions can always be backported to older schema versions. TODO - Specify how mapping errors can be reported to client.

8. Limitations of the solution

Not all schema conversions are possible. E.g. an impossible type conversion, or something has been removed. The solution is fundamentally limited by how the schemas actually change, this

solution does not provide a magic bullet that can solve all versioning issues.

9. Schema Version Selection YANG module

The YANG schema version selection YANG module is used by a device to report the schema-sets that are available, and to allow clients to choose which schema-set they wish to use.

Feature are used to allow servers to decide whether they allow the primary schema-set to be changed, and/or allow secondary schema-sets to be configured.

The primary schema-set is the datastore schema reported by YANG Library if a client connects to the device using the standard NETCONF/RESTCONF protocol numbers.

If secondary schema-sets are configured, then the client can choose whether NETCONF or RESTCONF is supported, which port numbers the protocols should run on (if available), and what RESTCONF root path prefix to use (e.g. if all of the RESTCONF protocol instances run on port 443).

Different schema-sets may support different datastores.

The "ietf-schema-version-selection" YANG module has the following structure:

```
module: ietf-schema-version-selection
  +--rw schema-selection
    +--rw schema-sets* [name]
      +--rw name string
      +--rw netconf! {secondary-schema-set}?
      | +--rw port? inet:port-number
      +--rw restconf! {secondary-schema-set}?
      | +--rw port? inet:port-number
      | +--rw root-path? inet:uri
      +--ro datastores* [datastore]
      | +--ro datastore ds:datastore-ref
      | +--ro package
      | | +--ro name?
      | | | -> /yanglib:yang-library/pkg:package/name
      | | +--ro version? leafref
      +--rw default-schema-set?
      | -> /schema-selection/schema-sets/name
      {default-schema-set}?
```

10. YANG Module

The YANG module definition for the module described in the previous sections.

```
<CODE BEGINS> file "ietf-schema-version-selection@2019-03-11.yang"
module ietf-schema-version-selection {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-schema-version-selection";
  prefix "ver-sel";

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference "RFC 8525: YANG Library";
  }
  import ietf-yang-library-packages {
    prefix pkg;
    reference "draft-rwilton-netmod-yang-packages-01";
  }

  organization
    "IETF NETMOD (Network Modeling) Working Group";

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

    Author:   Reshad Rahman
              <mailto:rrahman@cisco.com>

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

  description
    "This module provide a data model to advertise and allow the
    selection of schema versions by clients."
```

Copyright (c) 2019 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

```
// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.
revision 2019-03-11 {
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Schema Version Selection";
}

/*
 * Typedefs
 */

typedef yang-sem-ver {
  type string {
    pattern '\d+[\.]\d+[\.]\d+[mM]?';
  }
  description
    "Represents a YANG semantic version number.";
  reference
    "TODO - Should be defined by YANG versioning types module";
}

feature "default-schema-set" {
  description
    "Feature that allows clients to choose the default schema set
    to be used for clients that connect using the standard network
    configuration protocol port number or URL.

    Implementations may choose to only support this feature in
    <operational> to report the default-schema-set without
    allowing it to be configured.";
}
```

```
feature "secondary-schema-set" {
  description
    "Feature to choose if secondary schema sets may be configured
    by clients.

    Implementations may choose to only support this feature in
    <operational> to report secondary schema sets without
    allowing them to be configured.";
}

container schema-selection {
  description
    "YANG schema version selection";

  list schema-sets {
    key "name";

    description
      "All schema-sets that are available for client selection.";

    leaf name {
      type "string" {
        length "1..255";
      }
      description
        "The server assigned name of the schema-set.

        This should include the schema family, and appropriate
        versioning or release information";
    }
  }

  container netconf {
    if-feature "secondary-schema-set";

    presence "Make this schema-set available via NETCONF";
    description
      "NETCONF protocol settings for this schema set, if
      available";

    leaf port {
      type inet:port-number;
      description
        "The port numnber to use for interacting with this
        schema-set.  If not configured, then the port number is
        server allocated.";
      reference
        "RFC 6242: Using the NETCONF Protocol over SSH";
    }
  }
}
```

```
    }

    container restconf {
        if-feature "secondary-schema-set";

        presence
            "Make this schema-set available via RESTCONF";
        description
            "RESTCONF protocol settings for this schema set, if
            available";

        leaf port {
            type inet:port-number;
            default "443";
            description
                "The port numnber to use for interacting with this
                schema-set.  If not configured, then the port number
                defaults to the standard RESTCONF https port number of
                443";
            reference
                "RFC 8040: RESTCONF Protocol, section 2.1";
        }

        leaf root-path {
            type inet:uri;
            default "/restconf";
            description
                "The default root path to use to access the RESTCONF
                protocol instance for this schema-set";
        }
    }
}

list datastores {
    key "datastore";
    config false;

    description
        "The list of datastores supported for this schema set";

    leaf datastore {
        type ds:datastore-ref;
        description
            "The datastore that this datastore schema is associated
            with";
        reference
            "RFC 8342: Network Management Datastore Architecture
            (NMDA)";
    }
}
```

```
    }

    container package {
      description
        "YANG package associated with this datastore schema";

      leaf name {
        type leafref {
          path "/yanglib:yang-library/pkg:package/pkg:name";
        }
        description
          "The name of the YANG package this schema relates to";
      }
      leaf version {
        type leafref {
          path '/yanglib:yang-library/'
            + 'pkg:package[pkg:name = current()/../name]/'
            + 'pkg:version';
        }

        description
          "The version of the YANG package this schema relates
            to";
      }
    }
  }
}

leaf default-schema-set {
  if-feature "default-schema-set";
  type leafref {
    path '/schema-selection/schema-sets/name';
  }
  description
    "Specifies the default schema-set used by this device. This
      is the set of datastore schema that is used if a client
      connects using the standard protocol port numbers and URLs";
}
}
}
<CODE ENDS>
```

11. Security Considerations

To be defined.

12. IANA Considerations

TODO - Add registrations for YANG modules defined in this draft.

13. Open Questions/Issues

All issues, along with the draft text, are currently being tracked at: TODO - URL

14. Acknowledgements

The ideas that formed this draft are based on discussions with the YANG versioning design team, and other members of the NETMOD WG.

15. References

15.1. Normative References

[I-D.ietf-netconf-rfc7895bis]

Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-07 (work in progress), October 2018.

[I-D.ietf-netmod-module-tags]

Hopps, C., Berger, L., and D. Bogdanovic, "YANG Module Tags", draft-ietf-netmod-module-tags-07 (work in progress), March 2019.

[I-D.ietf-netmod-yang-instance-file-format]

Lengyel, B. and B. Claise, "YANG Instance Data File Format", draft-ietf-netmod-yang-instance-file-format-02 (work in progress), February 2019.

[I-D.rwilton-netmod-yang-packages]

Wilton, R., "YANG Packages", draft-rwilton-netmod-yang-packages-00 (work in progress), December 2018.

[I-D.verdt-netmod-yang-semver]

Claise, B., Clarke, J., Rahman, R., Wilton, R., Lengyel, B., Sterne, J., and K. D'Souza, "YANG Semantic Versioning for Modules", draft-verdt-netmod-yang-semver-00 (work in progress), March 2019.

[I-D.verdt-netmod-yang-versioning-reqs]

Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-02 (work in progress), November 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

15.2. Informative References

- [I-D.bierman-netmod-yang-package]
Bierman, A., "The YANG Package Statement", draft-bierman-netmod-yang-package-00 (work in progress), July 2015.

- [I-D.ietf-netmod-artwork-folding]
Watsen, K., Wu, Q., Farrel, A., and B. Claise, "Handling
Long Lines in Inclusions in Internet-Drafts and RFCs",
draft-ietf-netmod-artwork-folding-01 (work in progress),
March 2019.
- [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>>.

Authors' Addresses

Robert Wilton
Cisco Systems, Inc.

Email: rwilton@cisco.com

Reshad Rahman
Cisco Systems, Inc.

Email: rrahman@cisco.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2019

Q. Wu
R. Tao
R. Ranade
Huawei
June 29, 2019

NMDA Base Notification for Intent based configuration update
draft-wu-netmod-base-notification-nmda-03

Abstract

The Network Configuration Protocol (NETCONF) and RESTCONF provides mechanisms to manipulate configuration datastores. NMDA introduces additional datastores for systems that support more advanced processing chains converting configuration to operational state. However, client applications are not able to be aware of common events in these additional datastores of the management system, such as an intended configuration state change in NETCONF server or RESTCONF server, that may impact management applications, especially when a server is managed by multiple clients or management applications. This document defines a YANG module that allows a client to receive additional notifications for some common system events pertaining to the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

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 December 31, 2019.

Copyright Notice

Copyright (c) 2019 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
1.1. Terminology	3
2. NMDA Base Notifications for Intent based configuration Update	3
2.1. Overview	3
2.2. Data Model Design	5
2.3. Relation with NMDA Datastore Compare	6
2.4. Definitions	7
3. Security Considerations	13
4. IANA Considerations	13
5. Acknowledgements	14
6. Contributors	14
7. Normative References	14
Appendix A. Changes between revisions	15
Authors' Addresses	15

1. Introduction

The Network Configuration Protocol (NETCONF) [RFC6241] and RESTCONF [RFC8040] provides mechanisms to manipulate configuration datastores. NMDA introduces additional datastores (e.g., <intended>, <operational>) for systems that support more advanced processing chains converting configuration to operational state. However, client applications are not able to be aware of common events in those additional datastores of the management system, e.g., there are many background system activities (e.g., system internal interactions with hardware, interaction with protocols or other devices) that happen during propagation of a configuration change to the software and hardware components of a system. It is possible that some configuration could not be applied to <operational> due to either remnant Configuration, or missing resource, etc. There is a need for user or an application (configuration) to know the origin of failed configuration node and the reason why the configuration changes were not applied.

This document define a YANG module that allows a client to receive additional notifications for some common system events pertaining to

the Network Management Datastore Architecture (NMDA) defined in [RFC8342]. These notifications are designed to support the monitoring of the base system events within the server and not specific to any network management protocols such as NETCONF and RESTCONF.

The solution presented in this document is backwards compatible with [RFC6470].

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC8342] and are not redefined here:

- o operational state datastore
- o running configuration datastore
- o intended configuration datastore

2. NMDA Base Notifications for Intent based configuration Update

2.1. Overview

The YANG module in NETCONF Base Notifications [RFC6470] specifies the following 5 event notifications for the 'NETCONF' stream to notify a client application that the NETCONF server state has changed:

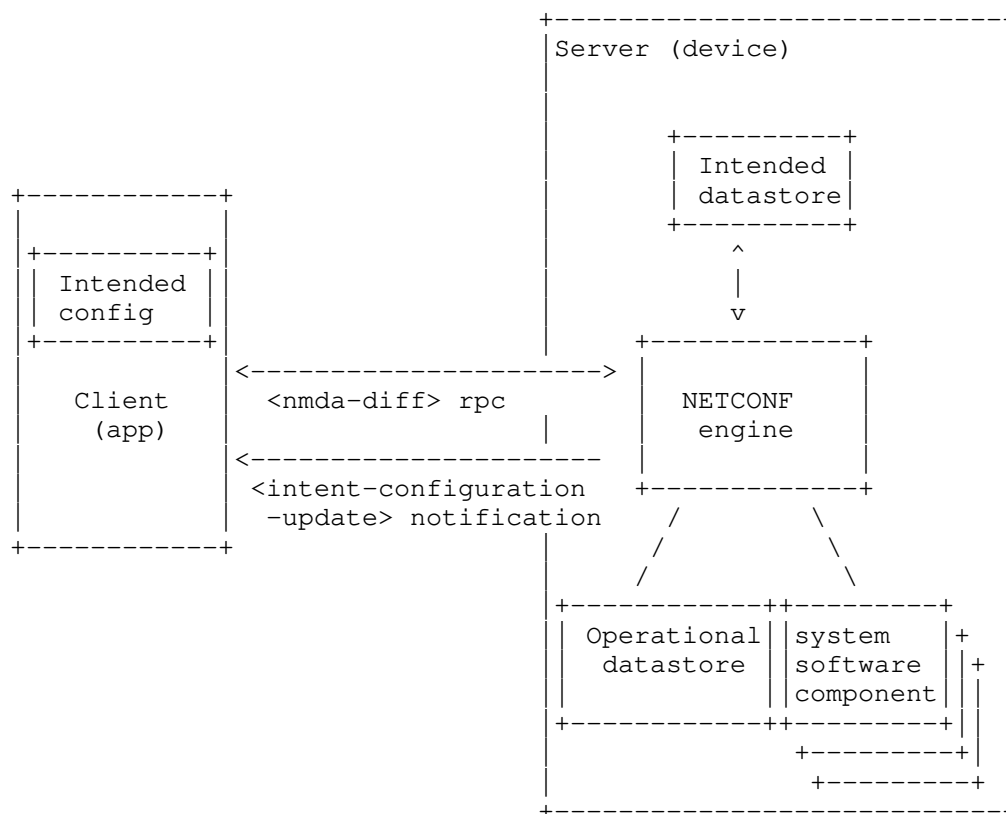
- o netconf-config-change
- o netconf-capability-change
- o netconf-session-start
- o netconf-session-end
- o netconf-confirmed-commit

These event notifications used within the 'NETCONF' stream are accessible to clients via the subscription mechanism described in [RFC5277].

This document introduces NMDA specific extension which allows a client to receive 1 notifications for additional common system events as follows:

apply-configuration-updated: Generated when a server with network management protocol support interacts with hardware and detects that a set of configurations are not applied or none of them are not applied. Indicates the event and the current state of the applied configuration or data inconsistency issue between intended data initiated from the client and operational data saved in the server. NMDA datastore compare [I-D.ietf-netmod-nmda-diff] can be used to trigger consistency data check, i.e., indicate the source of configuration node and check which part of configuration data is applied or which part of configuration data is not applied. A server MAY report events for non-NETCONF management sessions (such as RESTCONF,gPRC), using the 'session-id' value of zero.

The following figure shows event notification sequence defined in this document.



These notification messages are accessible to clients via either the subscription mechanism described in [RFC5277] or dynamic subscription mechanism and configured subscription mechanism described in [I-D.ietf-netconf-netconf-event-notifications].

2.2. Data Model Design

The data model is defined in the ietf-nmda-notifications YANG module. Its structure is shown in the following figure. The notation syntax follows [RFC8340].

```

notifications:
  +---n intent-configuration-update
    +--ro app-tag?          string
    +--ro src-ds?           identityref
    +--ro dst-ds?           identifyref
    +--ro (filter-spec)?
      | +---:(subtree-filter)
      | | +--ro subtree-filter?  <anydata>
      | +---:(xpath-filter)
      | | +--ro xpath-filter?    yang:xpath1.0 {nc:xpath}?
      | --ro apply-result        enumeration
    +--ro fail-applied-object* [edit-id]
      +--ro edit-id            string
      +--ro operation          enumeration
      +--ro object?            ypatch:target-resource-offset
      +--ro value?             <anydata>
      +--ro errors
        +--ro error* []
          +--ro error-type      enumeration
          +--ro error-tag       string
          +--ro error-app-tag?   string
          +--ro error-path?     instance-identifier
          +--ro error-message?   string
          +--ro error-info?     <anydata>

```

The following are examples of a apply-configuration-updated notification message:


```

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-06-16T16:30:59.137045+09:00</eventTime>
  <intent-configuration-update xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-notifications">
    <app-tag>ds--module-a</app-tag>
    <datastore>intended</datastore>
    <datastore>operational</datastore>
    <fail-applied-object>
      <edit-id>1</edit-id>
      <operation>merge</operation>
      <target>/ietf-interfaces:interfaces-state</target>
      <value>
        <interfaces-state xmlns="http://foo.com/ietf-interfaces">
          <interface>
            <name>eth0</name>
            <oper-status>down</oper-status>
          </interface>
        </interfaces-state>
      </value>
    </fail-applied-object>
    <fail-applied-object>
      <edit-id>2</edit-id>
      <target>/ietf-system:system</target>
      <errors>
        <error-type>protocol</error-type>
        <error-tag>mis-resource</error-tag>
        <error-path xmlns:ops="https://example.com/ns/ietf-system">\
          \if:interfaces-state\
        </error-path>
        <error-message>refer to resources that are not \
          \available or otherwise not physically present.\
        </error-message>
      </errors>
    </fail-applied-object>
  </intent-configuration-update>
</notification>

```

2.3. Relation with NMDA Datastore Compare

NMDA datastore compare [I-D.ietf-netmod-nmda-diff] could be used to check which part of configuration data is applied or which part of configuration data is not applied, e.g., If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, the interface will appear in <intended> but does not exist in the <operational>. By comparing configuration difference between <intended> and <operational>, the interface that is not applied can be sorted out. Unlike [I-D.ietf-netmod-nmda-diff], the notification message only focuses on the configuration data that is not applied and the reason why the configuration changes

were not applied. Also system internal interactions with hardware is needed within the server to make sure fail applied object is caused by mis-resource or remnant Configuration, etc.

2.4. Definitions

This section presents the YANG module defined in this document. This module imports data types from the 'ietf-datastores' module defined in [RFC8342] and 'ietf-inet-types' module defined in [RFC6021].

```
<CODE BEGINS> file "ietf-nmda-notifications@2019-06-19.yang"
module ietf-nmda-notifications {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-nmda-notifications";
  prefix ndn;
  import ietf-datastores {
    prefix ds;
  }
  import ietf-inet-types { prefix inet; }
  import ietf-yang-types { prefix yang; }
  import ietf-yang-patch {
    prefix ypatch;
  }
  import ietf-netconf {
    prefix nc;
  }
  import ietf-restconf { prefix rc; }
  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/netmod/>
    WG List:  <mailto:netmod@ietf.org>

    Editor:   Qin Wu
              <mailto:bill.wu@huawei.com>
    Editor:   Rohit R Ranade
              <mailto:rohitrranade@huawei.com>";
  description
    "This module defines a YANG data model for use with the
    NETCONF and RESTCONF protocol that allows the client to
    receive additional common event notifications related to NMDA.

    Copyright (c) 2012 IETF Trust and the persons identified as
    the document authors. All rights reserved.
    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
```

```
    Relating to IETF Documents
    (http://trustee.ietf.org/license-info).
    This version of this YANG module is part of RFC xxxx; see
    the RFC itself for full legal notices.";

revision 2019-06-19 {
  description
    "Initial version.";
  reference "RFC xxx: NETCONF Base Notifications for NMDA";
}
typedef session-id-or-zero-type {
  type uint32;
  description
    "NETCONF Session Id or Zero to indicate none";
}
feature error-info {
  description
    "This feature must
    also be enabled for that session if error-info
    can be advertised by the server. Otherwise,
    this feature must not be enabled.";
}
grouping common-session-parms {
  description
    "Common session parameters to identify a
    management session or internal interaction
    on a set of configuration data.";

  leaf username {
    type string;
    description
      "Name of the user for the session.";
  }
  leaf source-host {
    type inet:ip-address;
    description
      "Address of the remote host for the session.";
  }

  leaf session-id {
    type session-id-or-zero-type;
    description
      "Identifier of the session.
      A NETCONF session MUST be identified by a non-zero value.
      A non-NETCONF session MAY be identified by the value zero.";
  }
  leaf app-tag {
    type string;
  }
}
```

```
    description
      "The application tag used to identify the managment session
      or internal interaction on a set of configuration data.";
  }
}

notification intent-configuration-updated {
  description
    "Generated when a server detects that a
    intended configuration applied event has occurred. Indicates
    the event and the current state of the intended data applying
    procedure in progress.";
  reference "RFC 8342, Section 5";
  uses common-session-parms;
  leaf src-ds {
    type identityref {
      base ds:datastore;
    }
    description
      "Indicates which datastore is
      source of edit-data operation.";
  }
  leaf dst-ds {
    type identityref {
      base ds:datastore;
    }
    description
      "Indicates which datastore is
      target of edit-data operation.";
  }
  choice filter-spec {
    description
      "The content filter specification for this request.";
    anydata subtree-filter {
      description
        "This parameter identifies the portions of the
        target datastore to retrieve.";
      reference
        "RFC 6241: Network Configuration Protocol, Section 6.";
    }
    leaf xpath-filter {
      if-feature nc:xpath;
      type yang:xpath1.0;
      description
        "This parameter contains an XPath expression identifying
        the portions of the target datastore to retrieve."
    }
  }
}
```

If the expression returns a node-set, all nodes in the node-set are selected by the filter. Otherwise, if the expression does not return a node-set, then the get-data operation fails.

The expression is evaluated in the following XPath context:

- o The set of namespace declarations are those in scope on the 'xpath-filter' leaf element.
- o The set of variable bindings is empty.
- o The function library is the core function library, and the XPath functions defined in section 10 in RFC 7950.
- o The context node is the root node of the target datastore.";

```
    }
  }
leaf apply-result {
  type enumeration {
    enum "partial-fail" {
      description
        "A set of configuration data is not applied.";
    }
    enum "fail" {
      description
        "None of configuration data is applied.";
    }
    enum "sucess" {
      description
        "All configuration data is applied.";
    }
  }
}
description
  "Configuration data apply result.";
}
list fail-applied-object {
  when "../apply-result = 'partial-fail'";
  key edit-id;
  ordered-by user;
  leaf edit-id {
    type string;
    description
      "Response status is for the 'edit' list entry
      with this 'edit-id' value.";
```

```
    }
  leaf operation {
    type enumeration {
      enum create {
        description
          "The target data node is created using the supplied
          value, only if it does not already exist. The
          'target' leaf identifies the data node to be
          created, not the parent data node.";
      }
      enum delete {
        description
          "Delete the target node, only if the data resource
          currently exists; otherwise, return an error.";
      }

      enum insert {
        description
          "Insert the supplied value into a user-ordered
          list or leaf-list entry. The target node must
          represent a new data resource. If the 'where'
          parameter is set to 'before' or 'after', then
          the 'point' parameter identifies the insertion
          point for the target node.";
      }
      enum merge {
        description
          "The supplied value is merged with the target data
          node.";
      }
      enum move {
        description
          "Move the target node. Reorder a user-ordered
          list or leaf-list. The target node must represent
          an existing data resource. If the 'where' parameter
          is set to 'before' or 'after', then the 'point'
          parameter identifies the insertion point to move
          the target node.";
      }
      enum replace {
        description
          "The supplied value is used to replace the target
          data node.";
      }
      enum remove {
        description
          "Delete the target node if it currently exists.";
      }
    }
  }
}
```

```

    }
    mandatory true;
    description
      "The datastore operation requested for the associated
       'edit' entry.";
  }
  leaf target {
    type ypatch:target-resource-offset;
    description
      "Topmost node associated with the configuration change.
       A server SHOULD set this object to the node within
       the datastore that is being altered. A server MAY
       set this object to one of the ancestors of the actual
       node that was changed, or omit this object, if the
       exact node is not known.";
  }
  anydata value {
    description
      "Value used for this edit operation. The anydata 'value'
       contains the target resource associated with the
       'target' leaf.

       For example, suppose the target node is a YANG container
       named foo:

           container foo {
             leaf a { type string; }
             leaf b { type int32; }
           }

       The 'value' node contains one instance of foo:

           <value>
             <foo xmlns='example-foo-namespace'>
               <a>some value</a>
               <b>42</b>
             </foo>
           </value>

       ";
  }
  uses rc:errors {if-feature error-info;}
  description
    "List for fail applied objects that is not applied. ";
}
}
}
}
<CODE ENDS>

```

3. 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, defined in [RFC6242].

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:

/intent-configuration-update:

Event type itself indicates that intent based configuration has been updated. This event could alert an attacker that a datastore may have been altered.

/intent-configuration-updated/apply-result:

Indicates the specific intent based configuration update event state change that occurred. A value of 'success' probably indicates that intent based configuration has been applied successfully.

4. IANA Considerations

This document registers one XML namespace URN in the 'IETF XML registry', following the format defined in [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-nmda-notifications

Registrant Contact: The IESG.

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

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC7950]:

name: ietf-nmda-notifications

prefix: ndn

namespace: urn:ietf:params:xml:ns:yang:ietf-nmda-notifications

RFC: xxxx

5. Acknowledgements

Thanks to Juergen Schoenwaelder, Alex Clemm, Carey Timothy and Andy Berman, Sterne Jason to review this draft and Thank Xiaojian Ding provide important input to the initial version of this document.

6. Contributors

Chong Feng
Huawei
Email: frank.fengchong@huawei.com

7. Normative References

- [I-D.ietf-netmod-nmda-diff]
Clemm, A., Qu, Y., Tantsura, J., and A. Bierman,
"Comparison of NMDA datastores", draft-ietf-netmod-nmda-diff-01 (work in progress), May 2019.
- [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>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/info/rfc5277>>.
- [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>>.

- [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>>.
- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <<https://www.rfc-editor.org/info/rfc6470>>.
- [RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <<https://www.rfc-editor.org/info/rfc8072>>.
- [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>>.

Appendix A. Changes between revisions

v01 - v03

- o Change notification name into intent-configuration update.
- o Change title into NMDA Base event for intent based configuration update.
- o Clarify the usage of NMDA base event and relation with NMDA diff work.

v01 - v00

- o Add application tag support and use additional parameters to identify management session.
- o Remove apply-intended-start and apply-intended-end two notifications since they are not needed based on discussion.

Authors' Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Ran Tao
Huawei

Email: taoran20@huawei.com

Rohit R Ranade
Huawei
Divyashree Techno Park, Whitefield
Bangalore, Karnataka 560066
India

Email: rohitrranade@huawei.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2019

M. Wang
Q. Wu
Huawei
C. Xie
China Telecom
June 29, 2019

A YANG Data model for Policy based Event Management
draft-wwx-netmod-event-yang-02

Abstract

[RFC8328] defines a policy-based management framework that allow definition of a data model to be used to represent high-level, possibly network-wide policies. This document defines an YANG data model for the policy based event management [RFC7950]. The policy based Event YANG provides the ability for the network management function (within a controller, an orchestrator, or a network element) to control the configuration and monitor state change on the network element and take simple and instant action when a trigger condition on the system state is met.

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 December 31, 2019.

Copyright Notice

Copyright (c) 2019 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	2
2.1. Terminology	2
2.2. Tree Diagrams	3
3. Objectives	3
4. Relationship to YANG Push	4
5. Relationship to EVENT MIB	5
6. Model Overview	6
7. EVENT TRIGGER YANG Module	11
8. EVENT YANG Module	16
9. Security Considerations	21
10. IANA Considerations	22
11. Normative References	23
Appendix A. Usage Example of ECA model working with YANG PUSH .	24
Appendix B. Changes between revisions	26
Authors' Addresses	27

1. Introduction

[RFC8328] defines a policy-based management framework that allow definition of a data model to be used to represent high-level, possibly network-wide policies. This document defines an policy based Event Management YANG data model [RFC7950]. The policy based Event management YANG provides the ability for the network management function (within a controller, an orchestrator, or a network element) to control the configurations and monitor state parameters on the network element and take simple and instant action when a trigger condition on the system state is met.

The data model in this document is designed to be compliant with the Network Management Datastore Architecture (NMDA) [RFC8342].

2. Conventions used in this document

2.1. 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 [RFC2119]. In this

document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

This document uses the following terms:

Error A deviation of a system from normal operation [RFC3877].

Fault Lasting error or warning condition [RFC3877].

Event Something that happens which may be of interest or trigger the invocation of the rule. A fault, an alarm, a change in network state, network security threat, hardware malfunction, buffer utilization crossing a threshold, network connection setup, an external input to the system, for example [RFC3877].

2.2. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

3. Objectives

This section describes some of the design objectives for the policy based Event management Data Model:

- o The policy based Event management YANG should provide the ability for the network management function to control configuration and monitor state changes on a network element using the NETCONF/RESTCONF, and initiate simple actions whenever a trigger condition is met. For example, a NETCONF subscribed notification can be generated when a system state value exceeds the threshold.
- o Clear and precise identification of policy based Event types and managed objects.
- o Allow the server to inform the client that a certain Event is related to other Events.
- o Allow one event to be able to call another nested event.
- o The event data model defined in this document can be implemented on the management system that also implements EVENT-MIB; thus, the mapping between the event data model and ENTITY-MIB should be clear.

4. Relationship to YANG Push

YANG-push mechanism provides a subscription service for updates from a datastore. And it supports two types of subscriptions which are distinguished by how updates are triggered: periodic and on-change.

The On-change Push allow receivers to receive updates whenever changes to target managed objects occur. This document specifies a mechanism that provides three trigger conditions:

- o Existence: When a specific managed object appears, the trigger fires, e.g. reserved ports are configured.
- o Boolean: The user can set the type of boolean operator (e.g. unequal, equal, less, less-or-equal, greater, greater-or-equal, etc) and preconfigured threshold value (e.g. Pre-configured threshold). If the value of a managed object meet Boolean conditions, the trigger fires, e.g., when the boolean operator type is 'less', the trigger will be fired if the value of managed object is less than the pre-configured Boolean value.
- o Variation: The event that may be triggered when a managed object in multiple instances of the data tree is found and the current sampled value is either rising or falling and exceeds the pre-configured threshold, but the value at the last sampling interval does not exceed the pre-configured threshold. It also can be triggered when the current sample value change exceeds the pre-configured delta threshold. For example, when the 'startup' value is set to 'rising', the current sampled value of that managed object is greater than or equal to this threshold, and the value at the last sampling interval was less than this threshold, then one variation rising event is triggered for that managed object. In another example, if the 'startup' value is set to 'falling' and the system state value of the managed object is less than or equal to this threshold, and the value at the last sampling interval was greater than this threshold, then one variation falling event is triggered for that managed object.

And the YANG Push mechanism more focuses on the remote mirroring and monitoring of configuration and operational state. For example, for on change method, the subscriber will receive a notification if the change occurs. The model defined in this document provides a method which allow automatic adjusting the value of the corresponding managed object when some event is triggered. It establishes association between network service monitoring and network service provision and can use output generated by network service monitoring as input of network service provision and thereby provide automated

network management. The details of the usage example is described in Appendix A.

5. Relationship to EVENT MIB

If the device implements the EVENT-MIB [RFC2981], each entry in the `"/events/event/trigger"` list is mapped to `MteTriggerEntry`, `MteTriggerExistenceEntry`, `MteTriggerBooleanEntry`, `MteTriggerThresholdEntry`, `MteObjectsEntry`, `MteEventEntry`, `MteEventSetEntry`. respectively.

The following table lists the YANG data nodes with corresponding objects in the EVENT-MIB [RFC2981].

YANG data node in ietf-event.yang	EVENT-MIB Objects (RFC2981)
target	mteObjectsName
event-name	mteEventName
event-description	mteEventComment
value	mteEventSetValue
events/event/trigger/name	mteTriggerName
trigger-description	mteTriggerComment
frequency	mteTriggerFrequency
operator	mteTriggerBooleanComparison
value	mteTriggerBooleanValue
rising-event	mteTriggerThresholdRising
falling-event	mteTriggerThresholdFalling
delta-rising-event	mteTriggerThresholdDeltaRising
variation/startup	mteTriggerThresholdStartup
existence/enable	mteTriggerExistenceStartup
boolean/enable	mteTriggerBooleanStartup

6. Model Overview

The YANG data model for the Event management has been split into two modules:

- o The `ietf-event-trigger.yang` module defines a set of groupings for a generic trigger. It is intended that these groupings will be used by the policy based event management model or other models that require the trigger conditions. In this model, three trigger conditions are defined under the "test" choice node:
 - * **Existence:** When a specific managed object appears, the trigger fires.
 - * **Boolean:** The Boolean trigger condition is used to monitor the value of managed object. It compares the value of the managed object with the pre-configured threshold value and takes action according to the comparison result. The operator types include unequal, equal, less, less-or-equal, greater, and greater-or-equal. For example, if the operator is set to equal, an event is triggered when the value of the managed object equals the pre-configured value. The event will not be triggered again until the value becomes unequal and comes back to equal.
 - * **Variation:** A Variation trigger condition regularly compares the change value of the managed object with the delta threshold value. The Variation trigger condition can be used to monitor the value change of a managed object (when "statup" be set to 'delta-rising' or 'delta-falling'), if the change during the sample interval exceeds the delta threshold, the event is triggered. It can also be used to monitor the value variation of the managed object. In addition, if the value of the monitored object crosses a delta threshold multiple times in succession, the event is triggered only for the first crossing. For example, if the value of a sampled object crosses the rising threshold multiple times, only the first crossing triggers a rising alarm event.
- o The `ietf-event.yang` module defines four lists: trigger, target, event, and action. Triggers define the targets meeting some conditions that lead to events. Events trigger corresponding actions:
 - * Each trigger can be seen as a logical test that, if satisfied or evaluated to be true, cause the action to be carried out. The `ietf-event.yang` module uses groupings defined in `ietf-event-trigger.yang` to present the trigger attributes.

- * The target list defines managed objects that can be added to logging or be set to a new value on the trigger, the trigger test type, or the event that resulted in the actions.
- * The event list defines what happens when an event is triggered, i.e., trigger the corresponding action, e.g., adding a logging (i.e. Recording the triggered event), setting a value to the managed object or both. The group-id can be used to group a set of event that can be executed together, e.g., deliver a service or provide service assurance.
- * Nested-event are supported by allowing one event's trigger to reference other event's definitions using the call-event configuration. Called events apply their triggers and actions before returning to the calling event's trigger and resuming evaluation. If the called event is triggered, then it returns an effective boolean true value to the calling event. For the calling event, this is equivalent to a condition statement evaluating to a true value and evaluation of the event continues.
- * The action list consists of updates or invocations on local managed object attributes and defines a set of actions which will be performed (e.g. logging, set value, etc) when the corresponding event is triggered. The value to be set can use many variations on rule structure.

The following tree diagrams [RFC8340] provide an overview of the data model for "ietf-event-trigger" module and the "ietf-event" module.

```

module: ietf-event-trigger
  grouping existences-trigger
    +-- existences
      +-- target*      target
  grouping boolean-trigger
    +-- boolean
      +-- operator?    operator
      +-- value?       match-value
      +-- target*      target
  grouping variation-trigger
    +-- variation
      +-- rising-value?      match-value
      +-- rising-target*     target
      +-- falling-value?     match-value
      +-- falling-target*    target
      +-- delta-rising-value? match-value
      +-- delta-rising-target* target
      +-- delta-falling-value? match-value

```

```

    +-- delta-falling-target*    target
    +-- startup?                 enumeration
grouping trigger-grouping
+-- (test)?
+--:(existences)
|   +-- existences
|       +-- target*    target
+--:(boolean)
|   +-- boolean
|       +-- operator?    operator
|       +-- value?       match-value
|       +-- target*      target
+--:(variation)
    +-- variation
        +-- rising-value?        match-value
        +-- rising-target*       target
        +-- falling-value?       match-value
        +-- falling-target*      target
        +-- delta-rising-value?  match-value
        +-- delta-rising-target* target
        +-- delta-falling-value? match-value
        +-- delta-falling-target* target
        +-- startup?            enumeration

module: ietf-event
+--rw events
+--rw event* [event-name type]
+--rw event-name    string
+--rw type           identityref
+--rw event-description? string
+--rw group-id?     group-type
+--rw target*       trig:target
+--rw clear?        boolean
+--rw trigger* [name]
|   +--rw name                string
|   +--rw call-event?         -> ../../event-name
|   +--rw frequency
|       +--rw type?           identityref
|       +--rw periodic
|           +--rw interval    uint32
|           +--rw start?      yang:date-and-time
|           +--rw end?        yang:date-and-time
|       +--rw scheduling
|           +--rw month*      string
|           +--rw day-of-month* uint8
|           +--rw day-of-week* uint8
|           +--rw hour*       uint8
|           +--rw minute*     uint8

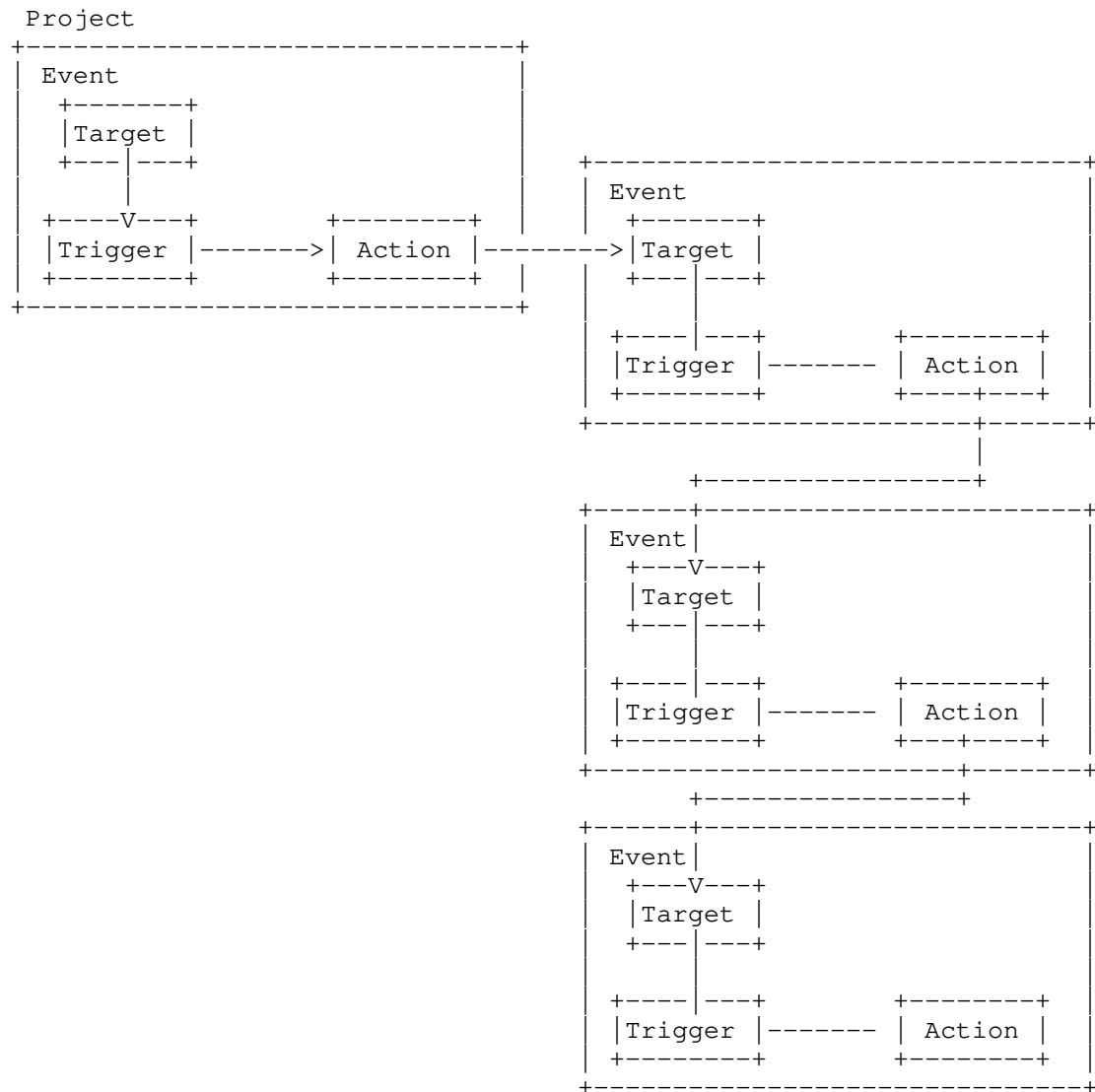
```

```

|      +---rw second*          uint8
|      +---rw start?           yang:date-and-time
|      +---rw end?             yang:date-and-time
+---rw (test)?
|   +---:(existences)
|   |   +---rw existences
|   |   |   +---rw target*      target
|   +---:(boolean)
|   |   +---rw boolean
|   |   |   +---rw operator?    operator
|   |   |   +---rw value?       match-value
|   |   |   +---rw target*      target
|   +---:(variation)
|   |   +---rw variation
|   |   |   +---rw rising-value?    match-value
|   |   |   +---rw rising-target*    target
|   |   |   +---rw falling-value?    match-value
|   |   |   +---rw falling-target*    target
|   |   |   +---rw delta-rising-value? match-value
|   |   |   +---rw delta-rising-target* target
|   |   |   +---rw delta-falling-value? match-value
|   |   |   +---rw delta-falling-target* target
|   |   |   +---rw startup?          enumeration
+---rw actions
|   +---rw target?      trig:target
|   +---rw value?       <anydata>
|   +---rw logging?     logging-type

```

The relation between Event, Trigger, Target and Action is described as follows:



One event may trigger another event, i.e., the action output in the first event can be input to target in the second event and a set of events can be grouped together and executed in a coordinated manner, but if it does not trigger another event, the relation between two events should be ignored.

7. EVENT TRIGGER YANG Module

```
<CODE BEGINS> file "ietf-event-trigger@2019-06-24.yang"
module ietf-event-trigger {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-event-trigger";
  prefix trig;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF xxx Working Group";
  contact
    "Zitao Wang: wangzitao@huawei.com
     Qin Wu: bill.wu@huawei.com";
  description
    "This module defines a reusable grouping for event trigger.";

  revision 2019-06-24 {
    description
      "Initial revision.";
    reference
      "foo";
  }

  typedef match-value {
    type union {
      type yang:xpath1.0;
      type yang:object-identifier;
      type string;
    }
    description
      "This type is used to match resources of type 'target'.
       Since the type 'target' is a union of different types,
       the 'match-value' type is also a union of corresponding
       types.";
  }

  typedef target {
    type union {
      type instance-identifier;
      type yang:object-identifier;
      type yang:uuid;
      type string;
    }
    description
```

```
"If the target is modelled in YANG, this type will
be an instance-identifier.
If the target is an SNMP object, the type will be an
object-identifier.
If the target is anything else, for example a distinguished
name or a CIM path, this type will be a string.
If the target is identified by a UUID use the uuid
type.
If the server supports several models, the presedence should
be in the order as given in the union definition.";
}

typedef operator {
  type enumeration {
    enum unequal {
      description
        "Indicates that the comparision type is unequal to.";
    }
    enum equal {
      description
        "Indicates that the comparision type is equal to.";
    }
    enum less {
      description
        "Indicates that the comparision type is less than.";
    }
    enum less-or-equal {
      description
        "Indicates that the comparision type is less than
        or equal to.";
    }
    enum greater {
      description
        "Indicates that the comparision type is greater than.";
    }
    enum greater-or-equal {
      description
        "Indicates that the comparision type is greater than
        or equal to.";
    }
  }
  description
    "definition of the operator";
}

grouping existences-trigger {
  description
    "A grouping that provides existence trigger";
```

```
    container existences {
      leaf-list target {
        type target;
        description
          "List for target objects";
      }
      description
        "Container for existence";
    }
  }

  grouping boolean-trigger {
    description
      "A grouping that provides boolean trigger";
    container boolean {
      leaf operator {
        type operator;
        description
          "Comparison type.";
      }
      leaf value {
        type match-value;
        description
          "Comparison value which is static threshold value.";
      }
      leaf-list target {
        type target;
        description
          "List for target management objects.";
      }
      description
        "Container for boolean test.";
    }
  }

  grouping variation-trigger {
    description
      "A grouping that provides variation trigger";
    container variation {
      leaf rising-value {
        type match-value;
        description
          "Sets the rising variation to the specified value,
           when the current sampled value is greater than or equal to
           this threshold, and the value at the last sampling interval
           was less than this threshold, the event is triggered. ";
      }
      leaf-list rising-target {
```



```
    type target;
    description
        "List for target objects.";
}
leaf falling-value {
    type match-value;
    description
        "Sets the falling threshold to the specified value.";
}
leaf-list falling-target {
    type target;
    description
        "List for target objects.";
}
leaf delta-rising-value {
    type match-value;
    description
        "Sets the delta rising threshold to the specified value.";
}
leaf-list delta-rising-target {
    type target;
    description
        "List for target objects.";
}
leaf delta-falling-value {
    type match-value;
    description
        "Sets the delta falling threshold to the specified value.";
}
leaf-list delta-falling-target {
    type target;
    description
        "List for target objects.";
}
leaf startup {
    type enumeration {
        enum rising {
            description
                "If the first sample after this
                 managed object becomes active is greater than or equal
                 to 'rising-value' and the 'startup' is equal to
                 'rising' then one threshold rising event is

                 triggered for that managed object.";
        }
        enum falling {
            description
                "If the first sample after this managed object becomes
```

```
        active is less than or equal to 'falling-value' and
        the 'startup' is equal to 'falling' then one
        threshold falling event is triggered for that managed
        object.";
    }
    enum rising-or-falling {
        description
            "That event may be triggered when the
            'startup' is equal to 'rising-or-falling'.
            'rising-or-falling' indicate the state value of the
            managed object may less than or greater than the
            specified threshold value.";
    }
}
description
    "Startup setting.";
}
description
    "Container for the threshold trigger condition.
    Note that the threshold here may change over time
    or the state value changes in either ascend order
    or descend order.";
}
}

grouping trigger-grouping {
    description
        "A grouping that provides event trigger.";
    choice test {
        description
            "Choice test";
        case existences {
            uses existences-trigger;
        }
        case boolean {
            uses boolean-trigger;
        }
        case variation {
            uses variation-trigger;
        }
    }
}
}
}
<CODE ENDS>
```

8. EVENT YANG Module

```
<CODE BEGINS> file "ietf-event@2019-06-24.yang"

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

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-event-trigger {
    prefix trig;
  }

  organization
    "IETF xxx Working Group";
  contact
    "Zitao Wang: wangzitao@huawei.com
     Qin Wu: bill.wu@huawei.com";
  description
    "This module defines a model for the service topology.";

  revision 2019-06-24 {
    description
      "Initial revision.";
    reference
      "foo";
  }

  identity event-type {
    description
      "Base identity for event type";
  }

  identity frequency {
    description
      "Base identity for frequency";
  }

  identity periodic {
    base frequency;
    description
      "Identity for periodic trigger";
  }

  identity scheduling {
```

```
    base frequency;
    description
        "Identity for scheduling trigger";
}

identity logging {
    description
        "Base identity for logging action";
}

identity logging-notification {
    base logging;
    description
        "Logging for event notification";
}

identity logging-set {
    base logging;
    description
        "Logging for reset values";
}

typedef logging-type {
    type identityref {
        base logging;
    }
    description
        "Logging types";
}

typedef group-type {
    type string;
    description
        "Group type";
}

grouping start-end-grouping {
    description
        "A grouping that provides start and end times for
        Event objects.";
    leaf start {
        type yang:date-and-time;
        description
            "The date and time when the Event object
            starts to create triggers.";
    }
    leaf end {
        type yang:date-and-time;
    }
}
```

```
    description
      "The date and time when the Event object
       stops to create triggers.
       It is generally a good idea to always configure
       an end time and to refresh the end time as needed
       to ensure that agents that lose connectivity to
       their Controller do not continue executing Schedules
       forever.";
  }
}

container events {
  list event {
    key "event-name type";
    leaf event-name {
      type string;
      description
        "Event name";
    }
    leaf type {
      type identityref {
        base event-type;
      }
      description
        "Type of event";
    }
    leaf event-description {
      type string;
      description
        "Event description";
    }
    leaf group-id {
      type group-type;
      description
        "Group Identifier";
    }
    leaf-list target {
      type trig:target;
      description
        "targeted objects";
    }
    leaf clear {
      type boolean;
      default "false";
      description
        "A flag indicate whether the event be closed";
    }
    list trigger {
```

```
key "name";
leaf name {
  type string;
  description
    "Trigger name";
}
leaf trigger-description {
  type string;
  description
    "Trigger description";
}
leaf call-event {
  type leafref {
    path "../../event-name";
  }
  description
    "This leaf call sub-event.";
}
container frequency {
  leaf type {
    type identityref {
      base frequency;
    }
    description
      "Type of trigger frequency";
  }
}
container periodic {
  when "derived-from-or-self(..../type, 'periodic')";
  description
    "A periodic timing object triggers periodically
    according to a regular interval.";
  leaf interval {
    type uint32 {
      range "1..max";
    }
    units "seconds";
    mandatory true;
    description
      "The number of seconds between two triggers
      generated by this periodic timing object.";
  }
  uses start-end-grouping;
}
container scheduling {
  when "derived-from-or-self(..../type, 'scheduling')";
  description
    "A scheduling timing object triggers.";
  leaf-list month {
```

```
    type string;
    description
        "A set of months at which this scheduling timing
        will trigger.";
}
leaf-list day-of-month {
    type uint8 {
        range "0..59";
    }
    description
        "A set of days of the month at which this
        scheduling timing will trigger.";
}
leaf-list day-of-week {
    type uint8 {
        range "0..59";
    }
    description
        "A set of weekdays at which this scheduling timing
        will trigger.";
}
leaf-list hour {
    type uint8 {
        range "0..59";
    }
    description
        "A set of hours at which the scheduling timing will
        trigger.";
}
leaf-list minute {
    type uint8 {
        range "0..59";
    }
    description
        "A set of minutes at which this scheduling timing
        will trigger.";
}
leaf-list second {
    type uint8 {
        range "0..59";
    }
    description
        "A set of seconds at which this calendar timing
        will trigger.";
}
uses start-end-grouping;
}
description
```

```
        "Container for frequency";
    }
    uses trig:trigger-grouping;
    description
        "List for trigger";
}
container actions {
    leaf target {
        type trig:target;
        description
            "Report the target objects";
    }
    anydata value {
        description
            "Inline set content.";
    }
    leaf logging {
        type logging-type;
        description
            "Specifies the log action";
    }
    description
        "Container for Actions";
}
description
    "List for Events";
}
description
    "YANG data module for defining event triggers and actions for
    network management purposes";
}
```

<CODE ENDS>

9. Security Considerations

The YANG modules defined in this document MAY be accessed via the RESTCONF protocol [RFC8040] or NETCONF protocol ([RFC6241]). The lowest RESTCONF or NETCONF layer requires that the transport-layer protocol provides both data integrity and confidentiality, see Section 2 in [RFC8040] and [RFC6241]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

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

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

- o /events/event/event-name
- o /events/event/target
- o /events/actions/target
- o /events/event/trigger/name

10. IANA Considerations

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

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

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

This document registers two YANG modules in the YANG Module Names registry [RFC6020].

```
-----
Name:          ietf-event-trigger
Namespace:     urn:ietf:params:xml:ns:yang:ietf-event-trigger
Prefix:        trig
Reference:     RFC xxxx

Name:          ietf-event
Namespace:     urn:ietf:params:xml:ns:yang:ietf-event
Prefix:        evt
Reference:     RFC xxxx
-----
```

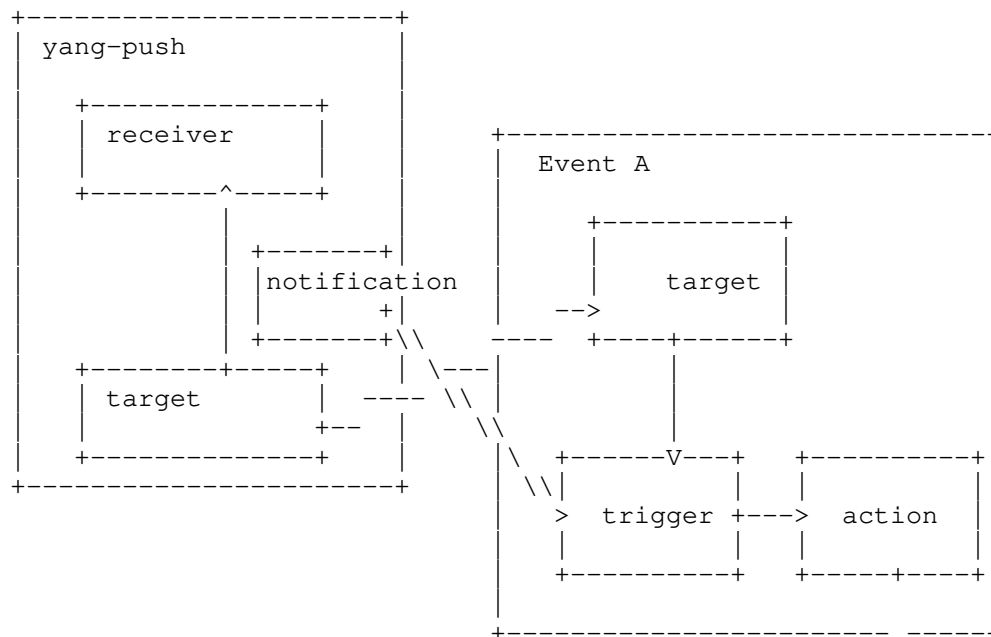
11. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.
- [RFC2981] Kavasseri, R., Ed., "Event MIB", RFC 2981, DOI 10.17487/RFC2981, October 2000, <<https://www.rfc-editor.org/info/rfc2981>>.
- [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>>.
- [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>>.
- [RFC6370] Bocci, M., Swallow, G., and E. Gray, "MPLS Transport Profile (MPLS-TP) Identifiers", RFC 6370, DOI 10.17487/RFC6370, September 2011, <<https://www.rfc-editor.org/info/rfc6370>>.
- [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>>.

- [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>>.
- [RFC8328] Liu, W., Xie, C., Strassner, J., Karagiannis, G., Klyus, M., Bi, J., Cheng, Y., and D. Zhang, "Policy-Based Management Framework for the Simplified Use of Policy Abstractions (SUPA)", RFC 8328, DOI 10.17487/RFC8328, March 2018, <<https://www.rfc-editor.org/info/rfc8328>>.

Appendix A. Usage Example of ECA model working with YANG PUSH

The relation between Event and YANG PUSH is described as follow: YANG Push Notification may trigger one event, i.e. one trigger conditions of the Event A can be set to "receiver received a yang push notification", and it can associates with other conditions of the Event A. When these conditions are met, the event A is triggered.



For Example:

The receiver received a push-change-update notification and learned that the "oper-status" of interface[name='eth0'] changed.

The target of Event "interface-state-monitoring" is set to "/if:interfaces/if:interface[if:name='eth0']", the trigger list contains two conditions: 1) receiver received a push-change-update notification; 2) the value of "in-errors" of interface[name='eth0'] exceeded the pre-configured threshold. When these conditions are met, corresponding action will be performed, i.e. disable interface[name='eth0']. The XML examples are shown as below:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-10-25T08:22:33.44Z</eventTime>
  <push-change-update
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>89</id>
    <datastore-changes>
      <yang-patch>
        <patch-id>0</patch-id>
        <edit>
          <edit-id>edit1</edit-id>
          <operation>replace</operation>
          <target>/ietf-interfaces:interfaces</target>
          <value>
            <interfaces
              xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
                <interface>
                  <name>eth0</name>
                  <oper-status>up</oper-status>
                </interface>
              </interfaces>
            </value>
          </edit>
        </yang-patch>
      </datastore-changes>
    </push-change-update>
  </notification>

  <event>
    <event-name>interface-state-monitoring</event-name>
    <type>interface-exception</type>
    <target>/if:interfaces/if:interface[if:name='eth0']</target>
    <trigger>
      <name>state-push-change</name>
```

```

    <trigger-description>received yang push
\changed notification</trigger-description>
    <test>
        <existence>/yp:notification/yp:push-change-update/yp:id[id=89]\
/yp:datastore-changes/.../yp:target="/ietf-interfaces:interfaces='eth0'\
"</existence>
    </test>
</trigger>
<trigger>
    <name>evaluate-in-errors</name>
    <call-event>interface-state-chang</call-event>
    <trigger-description>evaluate the number of
        the packets that contained errors
    </trigger-description>
    <frequency>10m</frequency>
    <test>
        <boolean>
            <operator>greater-or-equal</operator>
            <value>100</value>
            <target>/if:interfaces/if:interface[if:name='eth0']\
/if:statistic/if:in-errors</target>
        </boolean>
    </test>
</trigger>
<action>
    <target>/if:interfaces/if:interface[if:name='eth0']</target>
    <value>
        <interfaces>
            <interface>
                <name>eth0</name>
                <enable>>false</enable>
            </interface>
        </interfaces>
    </value>
</action>
</event>

</events>

```

Appendix B. Changes between revisions

v01 - v02

- o Introduce the group-id which allow group a set of events that can be executed together

- o Change threshold trigger condition into variation trigger condition to further clarify the difference between boolean trigger condition and variation trigger condition.
- o Module structure optimization.
- o Usage Example Update.

v00 - v01

- o Separate ietf-event-trigger.yang from Event management model and ietf-event.yang and make it reusable in other YANG models.
- o Clarify the difference between boolean trigger condition and threshold trigger condition.
- o Change evt-smp-min and evt-smp-max into min-data-object and max-data-object in the data model.

Authors' Addresses

Michael Wang
Huawei Technologies, Co., Ltd
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: wangzitao@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Chongfeng Xie
China Telecom

Email: xiechf@ctbri.com.cn