

Netmod WG
Internet-Draft
Intended status: Standards Track
Expires: 21 April 2022

O. Gonzalez de Dios
S. Barguil
Telefonica
M. Boucadair
Orange
18 October 2021

Extensions to the Access Control Lists (ACLs) YANG Model
draft-dbb-netmod-acl-00

Abstract

RFC 8519 defines a YANG data model for Access Control Lists (ACLs). This document discusses a set of extensions that fix many of the limitations of the ACL model as initially defined in RFC 8519.

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 21 April 2022.

Copyright Notice

Copyright (c) 2021 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. Approach	4
3. Problem Statement & Gap Analysis	4
3.1. Suboptimal Configuration: Lack of Manipulating Lists of Prefixes	4
3.2. Manageability: Impossibility to Use Aliases or Defined Sets	8
3.3. Bind ACLs to Devices, Not Only Interfaces	9
3.4. Partial or Lack of IPv4/IPv6 Fragment Handling	9
3.5. Suboptimal TCP Flags Handling	13
3.6. Rate-Limit Action	13
3.7. Payload-based Filtering	14
3.8. Reuse the ACLs Content Across Several Devices	15
4. Overall Module Structure (TBC)	15
5. YANG Module (TBC)	15
6. Security Considerations (TBC)	15
7. IANA Considerations	16
7.1. URI Registration (TBC)	16
7.2. YANG Module Name Registration (TBC)	16
8. Acknowledgements	16
9. Normative References	16
Authors' Addresses	17

1. Introduction

[RFC8519] defines Access control lists (ACLs) as a user-ordered set of filtering rules. The model targets the configuration of the filtering behaviour of a device. However, the model structure, as defined in [RFC8519], suffers from a set of limitations. This document describes these limitations and proposes an enhanced ACL structure.

The motivation of such enhanced ACL structure is discussed in detail in Section 3.

When managing ACLs, it is common for network operators to group matching elements in pre-defined sets. The consolidation into matches allows reducing the number of rules, especially in large scale networks. If it is needed, for example, to find a match against 100 IP addresses (or prefixes), a single rule will suffice rather than creating individual Access Control Entries (ACEs) for each IP address (or prefix). In doing so, implementations would optimize the performance of matching lists vs multiple rules matching.

The enhanced ACL structure is also meant to facilitate the management of network operators. Instead of entering the IP address or port number literals, using user-named lists decouples the creation of the rule from the management of the sets. Hence, it is possible to remove/add entries to the list without redefining the (parent) ACL rule.

In addition, the notion of Access Control List (ACL) and defined sets is generalized so that it is not device-specific as per [RFC8519]. ACLs and defined sets may be defined at network / administrative domain level and associated to devices. This approach facilitates the reusability across multiple network elements. For example, managing the IP prefix sets from a network level makes it easier to maintain by the security groups.

Network operators maintain sets of IP prefixes that are related to each other, e.g., deny-lists or accept-lists that are associated with those provided by a VPN customer. These lists are maintained and manipulated by security expert teams.

Note that ACLs are used locally in devices but are triggered by other tools such as DDoS mitigation [RFC9132] or BGP Flow Spec [RFC8955] [RFC8956]. Therefore, supporting means to easily map to the filtering rules conveyed in messages triggered by these tools is valuable from a network operation standpoint.

1.1. Terminology

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

The terminology for describing YANG modules is defined in [RFC7950]. The meaning of the symbols in the tree diagrams is defined in [RFC8340].

In addition to the terms defined in [RFC8519], this document makes use of the following terms:

- * Defined set: Refers to reusable description of one or multiple information elements (e.g., IP address, IP prefix, port number, ICMP type).

2. Approach

This first version of the document does not include on purpose any YANG module. This is because the authors are seeking a work direction from the netmod WG whether the missing features can be accomplished by means of augmentations or whether an ACL-bis document is more appropriate.

Future versions of the document will include a YANG module that will reflect the WG feedback. A network wide module, in addition to the device module, might be required. The decision on whether a single module is sufficient to handle both device and network levels or two separate ones will be based on WG feedback.

3. Problem Statement & Gap Analysis

3.1. Suboptimal Configuration: Lack of Manipulating Lists of Prefixes

IP prefix related data nodes, e.g., "destination-ipv4-network" or "destination-ipv6-network", do not allow manipulating a list of IP prefixes, which may lead to manipulating large files. The same issue is encountered when ACLs have to be in place to mitigate DDoS attacks (e.g., [RFC9132]) when a set of sources are involved in such an attack. The situation is even worse when both a list of sources and destination prefixes are involved.

Figure 1 shows an example of the required ACL configuration for filtering traffic from two prefixes.

```
{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "first-prefix",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "my-test-ace",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network":
                    "2001:db8:6401:1::/64",
                  "source-ipv6-network":
                    "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```

        "udp": {
            "source-port": {
                "operator": "lte",
                "port": 80
            },
            "destination-port": {
                "operator": "neq",
                "port": 1010
            }
        },
        "actions": {
            "forwarding": "accept"
        }
    }
}
},
{
    "name": "second-prefix",
    "type": "ipv6-acl-type",
    "aces": {
        "ace": [
            {
                "name": "my-test-ace",
                "matches": {
                    "ipv6": {
                        "destination-ipv6-network":
                            "2001:db8:6401:c::/64",
                        "source-ipv6-network":
                            "2001:db8:1234::/96",
                        "protocol": 17,
                        "flow-label": 10000
                    },
                    "udp": {
                        "source-port": {
                            "operator": "lte",
                            "port": 80
                        },
                        "destination-port": {
                            "operator": "neq",
                            "port": 1010
                        }
                    }
                },
                "actions": {
                    "forwarding": "accept"
                }
            }
        ]
    }
}

```

```
    }  
  ]  
}  
]  
}  
]  
}  
}
```

Figure 1: Example Illustrating Sub-optimal Use of the ACL Model
with a Prefix List.

Such configuration is suboptimal for both: - Network controllers that need to manipulate large files. All or a subset of this configuration will need to be passed to the underlying network devices. - Devices may receive such configuration and thus will need to maintain it locally.

Figure 2 depicts an example of an optimized structure:

```

{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "prefix-list-support",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "my-test-ace",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": [
                    "2001:db8:6401:1::/64",
                    "2001:db8:6401:c::/64"
                  ],
                  "source-ipv6-network":
                    "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}

```

Figure 2: Example Illustrating Optimal Use of the ACL Model in a Network Context.

3.2. Manageability: Impossibility to Use Aliases or Defined Sets

The same approach as the one discussed for IP prefixes can be generalized by introducing the concept of "aliases" or "defined sets".

The defined sets are reusable definitions across several ACLs. Each category is modelled in YANG as a list of parameters related to the class it represents. The following sets can be considered:

- * Prefix sets: Used to create lists of IPv4 or IPv6 prefixes.
- * Protocol sets: Used to create a list of protocols.
- * Port number sets: Used to create lists of TCP or UDP port values (or any other transport protocol that makes uses of port numbers). The identity of the protocols is identified by the protocol set, if present. Otherwise, a set apply to any protocol.
- * ICMP sets: Uses to create lists of ICMP-based filters. This applies only when the protocol is set to ICMP or ICMPv6.

A candidate structure is shown in #example_sets:

```

+--rw defined-sets
|   +--rw prefix-sets
|   |   +--rw prefix-set* [name mode]
|   |   |   +--rw name          string
|   |   |   +--rw mode          enumeration
|   |   |   +--rw ip-prefix*    inet:ip-prefix
|   +--rw port-sets
|   |   +--rw port-set* [name]
|   |   |   +--rw name          string
|   |   |   +--rw port*         inet:port-number
|   +--rw protocol-sets
|   |   +--rw protocol-set* [name]
|   |   |   +--rw name          string
|   |   |   +--rw protocol-name* identityref
|   +--rw icmp-type-sets
|   |   +--rw icmp-type-set* [name]
|   |   |   +--rw name          string
|   |   |   +--rw types* [type]
|   |   |   |   +--rw type          uint8
|   |   |   |   +--rw code?        uint8
|   |   |   |   +--rw rest-of-header? binary

```

Figure 3: Examples of Defined Sets.

3.3. Bind ACLs to Devices, Not Only Interfaces

In the context of network management, an ACL may be enforced in many network locations. As such, the ACL module should allow binding an ACL to multiple devices, not only (abstract) interfaces.

The ACL name must, thus, be unique at the scale of the network, but still the same name may be used in many devices when enforcing node-specific ACLs.

3.4. Partial or Lack of IPv4/IPv6 Fragment Handling

[RFC8519] does not support fragment handling capability for IPv6 but offers a partial support for IPv4 by means of 'flags'. Nevertheless, the use of 'flags' is problematic since it does not allow a bitmask to be defined. For example, setting other bits not covered by the 'flags' filtering clause in a packet will allow that packet to get through (because it won't match the ACE).

Defining a new IPv4/IPv6 matching field called 'fragment' is thus required to efficiently handle fragment-related filtering rules. Some examples to illustrate how 'fragment' can be used are provided below.

Figure 4 shows the content of a candidate POST request to allow the traffic destined to 198.51.100.0/24 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

- * "drop-all-fragments" ACE: discards all fragments.
- * "allow-dns-packets" ACE: accepts DNS packets destined to 198.51.100.0/24.

```

{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "dns-fragments",
        "type": "ipv4-acl-type",
        "aces": {
          "ace": [
            {
              "name": "drop-all-fragments",
              "matches": {
                "ipv4": {
                  "fragment": {
                    "operator": "match",
                    "type": "isf"
                  }
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            },
            {
              "name": "allow-dns-packets",
              "matches": {
                "ipv4": {
                  "destination-ipv4-network": "198.51.100.0/24"
                },
                "udp": {
                  "destination-port": {
                    "operator": "eq",
                    "port": 53
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}

```

Figure 4: Example Illustrating Candidate Filtering of IPv4 Fragmented Packets.

Figure 5 shows an example of the body of a candidate POST request to allow the traffic destined to 2001:db8::/32 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

- * "drop-all-fragments" ACE: discards all fragments (including atomic fragments). That is, IPv6 packets that include a Fragment header (44) are dropped.
- * "allow-dns-packets" ACE: accepts DNS packets destined to 2001:db8::/32.

```

{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "dns-fragments",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "drop-all-fragments",
              "matches": {
                "ipv6": {
                  "fragment": {
                    "operator": "match",
                    "type": "isf"
                  }
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            },
            {
              "name": "allow-dns-packets",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": "2001:db8::/32"
                },
                "udp": {
                  "destination-port": {
                    "operator": "eq",
                    "port": 53
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}

```

Figure 5: Example Illustrating Candidate Filtering of IPv6 Fragmented Packets.

3.5. Suboptimal TCP Flags Handling

[RFC8519] allows including flags in the TCP match fields, however that structure does not support matching operations as those supported in BGP Flow Spec. Defining this field to be defined as a flag bitmask together with a set of operations is meant to efficiently handle TCP flags filtering rules. Some examples to illustrate the use of such field are discussed below.

Figure 6 shows an example of a candidate request to install a filter to discard incoming TCP messages having all flags unset.

```
{
  "ietf-access-control-list:acls": {
    "acl": [{
      "name": "tcp-flags-example",
      "aces": {
        "ace": [{
          "name": "null-attack",
          "matches": {
            "tcp": {
              "flags-bitmask": {
                "operator": "not any",
                "bitmask": 4095
              }
            }
          },
          "actions": {
            "forwarding": "drop"
          }
        }]
      }
    }]
  }
}
```

Figure 6: Example to Deny TCP Null Attack Messages

3.6. Rate-Limit Action

[RFC8519] specifies that forwarding actions can be 'accept' (i.e., accept matching traffic), 'drop' (i.e., drop matching traffic without sending any ICMP error message), or 'reject' (i.e., drop matching traffic and send an ICMP error message to the source). However, there are situations where the matching traffic can be accepted, but with a rate-limit policy. Such capability is not currently supported by the ACL model.

Figure 7 shows a candidate ACL example to rate-limit incoming SYNs during a SYN flood attack.

```
{
  "ietf-access-control-list:acls": {
    "acl": [{
      "name": "tcp-flags-example-with-rate-limit",
      "aces": {
        "ace": [{
          "name": "rate-limit-syn",
          "matches": {
            "tcp": {
              "flags-bitmask": {
                "operator": "match",
                "bitmask": 2
              }
            }
          },
          "actions": {
            "forwarding": "accept",
            "rate-limit": "20.00"
          }
        }]
      }
    }]
  }
}
```

Figure 7: Example Rate-Limit Incoming TCP SYNs

3.7. Payload-based Filtering

Some transport protocols use existing protocols (e.g., TCP or UDP) as substrate. The match criteria for such protocols may rely upon the 'protocol' under 'l3', TCP/UDP match criteria, part of the TCP/UDP payload, or a combination thereof. [RFC8519] does not support matching based on the payload.

Likewise, the current version of the ACL model does not support filetering of encapsulated traffic.

3.8. Reuse the ACLs Content Across Several Devices

Having a global network view of the ACLs is highly valuable for service providers. An ACL could be defined and applied following the hierarchy of the network topology. So, an ACL can be defined at the network level and, then, that same ACL can be used (or referenced to) in several devices (including termination points) within the same network.

This network/device ACLs differentiation introduces several new requirements, e.g.:

- * An ACL name can be used at both network and device levels.
- * An ACL content updated at the network level should imply a transaction that updates the relevant content in all the nodes using this ACL.
- * ACLs defined at the device level have a local meaning for the specific node.
- * A device can be associated with a router, a VRF, a logical system, or a virtual node. ACLs can be applied in physical and logical infrastructure.

4. Overall Module Structure (TBC)

To be completed.

5. YANG Module (TBC)

To be completed.

6. Security Considerations (TBC)

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocol 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 Network Configuration Access Control Model (NACM) [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.

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:

* TBC

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:

* TBC

7. IANA Considerations

7.1. URI Registration (TBC)

This document requests IANA to register the following URI in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:xxx
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

7.2. YANG Module Name Registration (TBC)

This document requests IANA to register the following YANG module in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

name: xxxx
namespace: urn:ietf:params:xml:ns:yang:ietf-xxx
maintained by IANA: N
prefix: xxxx
reference: RFC XXXX

8. Acknowledgements

Many thanks to Jon Shallow and Miguel Cros for the discussion when preparing this draft.

9. 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <<https://www.rfc-editor.org/info/rfc8519>>.

Authors' Addresses

Oscar Gonzalez de Dios
Telefonica
Distrito T
Madrid

Email: oscar.gonzalezdedios@telefonica.com

Samier Barguil
Telefonica
Distrito T
Madrid

Email: samier.barguilgiraldo.ext@telefonica.com

Mohamed Boucadair
Orange
Rennes

Email: mohamed.boucadair@orange.com

Network Working Group
Internet-Draft
Updates: 7950,8407,8525 (if approved)
Intended status: Standards Track
Expires: January 13, 2022

R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman, Ed.
B. Lengyel, Ed.
Ericsson
J. Clarke
Cisco Systems, Inc.
J. Sterne
Nokia
July 12, 2021

Updated YANG Module Revision Handling
draft-ietf-netmod-yang-module-versioning-03

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. It provides a mechanism, via the revision-label YANG extension, to specify a revision identifier for YANG modules and submodules. 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 13, 2022.

Copyright Notice

Copyright (c) 2021 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	5
3.1. Updating a YANG module with a new revision	6
3.1.1. Backwards-compatible rules	6
3.1.2. Non-backwards-compatible changes	7
3.2. non-backwards-compatible revision extension statement . .	7
3.3. Removing revisions from the revision history	7
3.4. Revision label	8
3.4.1. File names	8
3.4.2. Revision label scheme extension statement	9
3.5. Examples for updating the YANG module revision history .	9
4. Import by derived revision	12
4.1. Module import examples	14
5. Updates to ietf-yang-library	15
5.1. Resolving ambiguous module imports	15
5.2. YANG library versioning augmentations	16
5.2.1. Advertising revision-label	16
5.2.2. Reporting how deprecated and obsolete nodes are handled	16
6. Versioning of YANG instance data	17
7. Guidelines for using the YANG module update rules	17
7.1. Guidelines for YANG module authors	17
7.1.1. Making non-backwards-compatible changes to a YANG module	18
7.2. Versioning Considerations for Clients	20
8. Module Versioning Extension YANG Modules	20
9. Contributors	29
10. Security Considerations	30
11. IANA Considerations	30

11.1. YANG Module Registrations	30
11.2. Guidance for versioning in IANA maintained YANG modules	31
12. References	32
12.1. Normative References	32
12.2. Informative References	33
Appendix A. Examples of changes that are NBC	34
Appendix B. Examples of applying the NBC change guidelines	35
B.1. Removing a data node	35
B.2. Changing the type of a leaf node	36
B.3. Reducing the range of a leaf node	37
B.4. Changing the key of a list	37
B.5. Renaming a node	38
B.6. Changing a default value	39
Appendix C. Changes between revisions	39
Authors' Addresses	39

1. Introduction

This document defines a solution to the YANG module lifecycle problems described in [I-D.ietf-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.ietf-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 document 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 and submodule descriptions, to report how "deprecated" and "obsolete" nodes are handled by a server, and to clarify how module imports are resolved when multiple revisions 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.

Note to RFC Editor (To be removed by RFC Editor)

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 [RFC7950] section 5.2. Section 3.4.1 describes the use of a revision label in the name of a file containing a YANG module or submodule.

This document updates [RFC7950] section 5.6.5. Section 5.1 defines how a client of a YANG library datastore schema resolves ambiguous imports for modules which are not "import-only".

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.

This document updates [RFC8525] with augmentations to include revision labels in the YANG library data and two boolean leaves to indicate whether status deprecated and status obsolete schema nodes are implemented by the server.

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 or submodule is strictly linear, i.e., it is prohibited to have two independent revisions of a YANG module or submodule that are both directly derived from the same parent revision.

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

A corollary to the above is that the relationship between two module or submodule revisions cannot be determined by comparing the module or submodule revision date alone, and the revision history, or revision label, must also be taken into consideration.

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 to ensure that the module's content is uniquely defined, the module's "include" statements SHOULD use "revision-date" substatements to specify the exact revision date of each included submodule. When a module does not include its submodules by revision-date, the revision of submodules used cannot be derived from the including module. Mechanisms such as YANG packages [I-D.ietf-netmod-yang-packages], and YANG library [RFC7895] [RFC8525], MAY be used to specify the exact submodule revisions used when the submodule revision date is not constrained by the "include" statement.

[RFC7950] section 11 requires that all updates to a YANG module are BC to the previous revision of the module. This document introduces a method to indicate that an NBC change has occurred between module revisions: this is done by using a new "non-backwards-compatible" YANG extension statement in the module revision history.

Two revisions of a module or submodule MAY have identical content except for the revision history. This could occur, for example, if a module or submodule has a branched history and identical changes are applied in multiple branches.

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, e.g., the semantics of an existing data-node definition MAY be changed in an NBC manner without requiring a new data-node definition with a new identifier. A YANG extension, defined in Section 3.2, is used to signal the potential for incompatibility to existing module users and readers.

As per [RFC7950], all published revisions of a module are given a new unique revision date. This applies even for module revisions containing (in the module or included submodules) only changes to any whitespace, formatting, comments or line endings (e.g., DOS vs UNIX).

3.1.1. Backwards-compatible rules

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 YANG schema nodes with a "status" "obsolete" substatement 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 of any "input" or "output" data definition substatements of "rpc" or "action" statements. If new data definition statements are added, they can be added anywhere in the sequence of existing substatements.

- o Any changes (including whitespace or formatting changes) that do not change the semantic meaning of the module are backwards compatible.
- o A statement that is defined using the YANG "extension" statement MAY be added, removed, or changed, if it does not change the semantics of the module. Extension statement definitions SHOULD specify whether adding, removing, or changing statements defined by that extension are backwards-compatible or non-backwards-compatible.

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. non-backwards-compatible revision extension statement

The "rev:non-backwards-compatible" 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:non-backwards-compatible" extension statement MUST be added as a substatement to the "revision" statement.

3.3. Removing revisions from the revision history

Authors may wish to remove revision statements from a module or submodule. Removal of revision information may be desired for a number of reasons including reducing the size of a large revision history, or removing a revision that should no longer be used or imported. Removing revision statements is allowed, but can cause issues and SHOULD NOT be done without careful analysis of the potential impact to users of the module or submodule. Doing so can lead to import breakages when import by revision-or-derived is used. Moreover, truncating history may cause loss of visibility of when non-backwards-compatible changes were introduced.

If a revision containing a rev:non-backwards-compatible substatement is removed from the revision history then a rev:non-backwards-compatible substatement MUST be added to the nearest newer revision entry in the revision history that is not being removed.

3.4. 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.

YANG Semver [I-D.ietf-netmod-yang-semver] defines a versioning scheme based on Semver 2.0.0 [semver] that can be used as a revision label.

Submodules MAY use a revision label scheme. When they use a revision label scheme, submodules MAY use a revision label scheme that is different from the one used in the including module.

The revision label space of submodules is separate from the revision label space of the including module. A change in one submodule MUST result in a new revision label of that submodule and the including module, but the actual values of the revision labels in the module and submodule could be completely different. A change in one submodule does not result in a new revision label in another submodule. A change in a module revision label does not necessarily mean a change to the revision label in all included submodules.

If a revision has an associated revision label, then it may be used instead of the revision date in a "rev:revision-or-derived" extension statement argument.

A specific revision-label identifies a specific revision (variant) of the module. If two YANG modules contain the same module name and the same revision-label (and hence also the same revision-date) in their latest revision statement, then the file contents of the two modules, including the revision history, MUST be identical.

3.4.1. File names

This section updates [RFC7950] section 5.2.

If a revision has an associated revision label, then the revision-label may be used instead of the revision date in the filename of a YANG file, where it takes the form:

```
module-or-submodule-name [['@' revision-date] | ['#' revision-label]]  
    ( '.yang' / '.yin' )
```

E.g., acme-router-module@2018-01-25.yang

E.g., acme-router-module#2.0.3.yang

YANG module (or submodule) files MAY be identified using either revision-date or revision-label. Typically, only one file name SHOULD exist for the same module (or submodule) revision. Two file names, one with the revision date and another with the revision label, MAY exist for the same module (or submodule) revision, e.g., when migrating from one scheme to the other.

3.4.2. Revision label scheme extension statement

The optional "rev:revision-label-scheme" extension statement is used to indicate which revision-label scheme a module or submodule uses. There MUST NOT be more than one revision label scheme in a module or submodule. The mandatory argument to this extension statement:

- o specifies the revision-label scheme used by the module or submodule
- o is defined in the document which specifies the revision-label scheme
- o MUST be an identity derived from "revision-label-scheme-base".

The revision-label scheme used by a module or submodule SHOULD NOT change during the lifetime of the module or submodule. If the revision-label scheme used by a module or submodule is changed to a new scheme, then all revision-label statements that do not conform to the new scheme MUST be replaced or removed.

3.5. Examples for updating the YANG module revision history

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

Example YANG module with branched revision history.

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

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

Example module, revision 2019-06-01:

```
module example-module {  
  
    namespace "urn:example:module";  
    prefix "prefix-name";  
    rev:revision-label-scheme "yangver:yang-semver";  
  
    import ietf-yang-revisions { prefix "rev"; }  
    import ietf-yang-semver { prefix "yangver"; }  
  
    description  
        "to be completed";  
  
    revision 2019-06-01 {  
        rev:revision-label 3.1.0;  
        description "Add new functionality.";  
    }  
  
    revision 2019-03-01 {  
        rev:revision-label 3.0.0;  
        rev:non-backwards-compatible;  
        description  
            "Add new functionality. Remove some deprecated nodes.";  
    }  
  
    revision 2019-02-01 {  
        rev:revision-label 2.0.0;  
        rev:non-backwards-compatible;  
        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 "urn:example:module";  
    prefix "prefix-name";  
    rev:revision-label-scheme "yangver:yang-semver";  
  
    import ietf-yang-revisions { prefix "rev"; }  
    import ietf-yang-semver { prefix "yangver"; }  
  
    description  
        "to be completed";  
  
    revision 2019-05-01 {  
        rev:revision-label 2.2.0;  
        description "Backwards-compatible bugfix to enhancement.";  
    }  
  
    revision 2019-04-01 {  
        rev:revision-label 2.1.0;  
        description "Apply enhancement to older release train.";  
    }  
  
    revision 2019-02-01 {  
        rev:revision-label 2.0.0;  
        rev:non-backwards-compatible;  
        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 an 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 accommodation those changes, along with updated import "revision-or-derived" extension statements to depend on the updated imported module revision.

Adding, modifying or removing a "revision-or-derived" extension statement is considered to be a BC change.

Adding, modifying or removing a "revision-date" extension statement is considered to be a BC change.

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 {
  rev:revision-or-derived 2019-02-01;
}
```

Alternatively, the first example could have used the revision label "2.0.0" instead, which selects the same set of revisions/labels.

```
import example-module {
  rev:revision-or-derived 2.0.0;
}
```

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 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 {  
  rev: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 {  
  rev:revision-or-derived 2019-04-01;  
  rev:revision-or-derived 2019-06-01;  
}
```

5. Updates to ietf-yang-library

This document updates YANG library [RFC7950] to clarify how ambiguous module imports are resolved. It also defines the YANG module, `ietf-yang-library-revisions` that augments YANG library [RFC8525] with new revision-label related meta-data.

5.1. 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.

5.2. YANG library versioning augmentations

The "ietf-yang-library-revisions" YANG module has the following structure (using the notation defined in [RFC8340]):

```
module: ietf-yang-library-revisions
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module
    /yanglib:submodule:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:module-set
    /yanglib:import-only-module/yanglib:submodule:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:schema:
    +--ro deprecated-nodes-implemented?   boolean
    +--ro obsolete-nodes-absent?          boolean
```

5.2.1. Advertising revision-label

The ietf-yang-library-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.

5.2.2. Reporting how deprecated and obsolete nodes are handled

The ietf-yang-library-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 set to "true", 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 set to "false" or absent, then the behavior is unspecified.

obsolete-nodes-absent: If set to "true", this leaf indicates that the server does not implement any status "obsolete" nodes. If

this leaf is set to "false" or absent, then the behaviour is unspecified.

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

If a server does not set the "deprecated-nodes-implemented" leaf to "true", then clients MUST NOT rely solely on the "rev:non-backwards-compatible" 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.ietf-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

All IETF YANG modules MUST include revision-label statements for all newly published YANG modules, and all newly published revisions of existing YANG modules. The revision-label MUST take the form of a YANG semantic version number [I-D.ietf-netmod-yang-semver].

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:non-backwards-compatible" statement MUST be added 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 to remove, or curtail, the older description statements associated with the previous revisions.

The "rev: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 SHOULD use the "revision-date" statement to include specific submodule revisions. The revision of the including module MUST be updated when any included submodule has changed. The revision-label substatement used in the new module revision MUST indicate the nature of the change, i.e. NBC or BC, to the module's schema tree.

In some cases a module or submodule revision that is not strictly NBC by the definition in Section 3.1.2 of this specification may include the "non-backwards-compatible" statement. Here is an example when adding the statement may be desirable:

- o A "config false" leaf had its value space expanded (for example, a range was increased, or additional enum values were added) and the author or server implementor feels there is a significant compatibility impact for clients and users of the module or submodule

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 revisions of the YANG module or of YANG packages(as specified in [I-D.ietf-netmod-yang-packages]), and allows the client to select the revision (as per [I-D.ietf-netmod-yang-ver-selection]), then NBC changes MAY be done without using "status" statements. Clients would be required to select the revision 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. For deprecated data nodes the "description" statement 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 "description". The reason for deprecating the node can also be included in the "description" if it is deemed to be of potential interest to the user.
3. For obsolete data nodes, it is RECOMMENDED to keep the above information, from when the node had status "deprecated", which is still relevant.
4. When obsoleting or deprecating data nodes, the "deprecated" or "obsolete" status SHOULD be applied at the highest possible level in the data tree. For clarity, the "status" statement SHOULD also be applied to all descendent data nodes, but the additional status related information does not need to be repeated if it does not introduce any additional information.
5. NBC changes which can break imports SHOULD be avoided because of the impact on the importing module. The importing modules could get broken, e.g., if an augmented node in the importing module has been removed from the imported module. Alternatively, the schema of the importing modules could undergo an NBC change due

to the NBC change in the imported module, e.g., if a node in a grouping has been removed. As described in Appendix B.1, instead of removing a node, that node SHOULD first be deprecated and then obsoleted.

See Appendix B 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, revision label scheme, and importing by revision.

```
<CODE BEGINS> file "ietf-yang-revisions@2021-06-30.yang"
module ietf-yang-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-revisions";
  prefix rev;

  // RFC Ed.: We need the bis version to get the new type revision-identifier
  // If 6991-bis is not yet an RFC we need to copy the definition here
  import ietf-yang-types {
    prefix yang;
    reference
      "XXXX [ietf-netmod-rfc6991-bis]: Common YANG Data Types";
  }
}
```

organization

"IETF NETMOD (Network Modeling) Working Group";

contact

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

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

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

Author: Reshad Rahman
<<mailto:reshad@yahoo.com>>

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

Copyright (c) 2021 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.

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.";

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

```
revision 2021-06-30 {
  description
    "Initial version.";
  reference
    "XXXX: Updated YANG Module Revision Handling";
}

typedef revision-label {
  type string {
    length "1..255";
    pattern '[a-zA-Z0-9,\-_\.]+';
    pattern '\d{4}-\d{2}-\d{2}' {
      modifier invert-match;
    }
  }
  description
    "A label associated with a YANG revision.

    Alphanumeric characters, comma, hyphen, underscore, period
    and plus are the only accepted characters. MUST NOT match
    revision-date.";
  reference
    "XXXX: Updated YANG Module Revision Handling;
    Section 3.3, Revision label";
}

typedef revision-date-or-label {
  type union {
    type yang:revision-identifier;
    type revision-label;
  }
  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.

    The statement MUST only be a substatement of the 'revision'
    statement. Zero or one 'non-backwards-compatible' statements
    per parent statement is allowed. No substatements for this
    extension have been standardized.

    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 'non-backwards-compatible' statement MUST be added as a substatement to the revision statement.

Conversely, if a revision does not contain a 'non-backwards-compatible' statement then all changes, relative to the preceding revision in the revision history, MUST be backwards-compatible.

A new module revision that only contains changes that are backwards compatible SHOULD NOT include the 'non-backwards-compatible' statement. An example of when an author might add the 'non-backwards-compatible' statement is if they believe a change could negatively impact clients even though the backwards compatibility rules defined in RFC-XXXX classify it as a backwards-compatible change.

Add, removing, or changing a 'non-backwards-compatible' statement is a backwards-compatible version change.";

reference

"XXXX: Updated YANG Module Revision Handling;
Section 3.2, nbc-changes revision extension statement";

}

extension revision-label {
 argument revision-label;
 description

"The revision label can be used to provide an additional versioning identifier associated with a module or submodule revision. E.g., one option for a versioning scheme that could be used is [XXXX: ietf-netmod-yang-semver].

The format of the revision-label argument MUST conform to the pattern defined for the revision-label typedef.

The statement MUST only be a substatement of the revision statement. Zero or one revision-label statements per parent statement are allowed. No substatements for this extension have been standardized.

Revision labels MUST be unique amongst all revisions of a module or submodule.

Adding a revision label is a backwards-compatible version change. Changing or removing an existing revision label in the revision history is a non-backwards-compatible version change, because it could impact any references to that revision label.";

```
reference
  "XXXX: Updated YANG Module Revision Handling;
  Section 3.3, Revision label";
}

extension revision-label-scheme {
  argument revision-label-scheme-identity;
  description
    "The revision label scheme specifies which revision-label scheme
    the module or submodule uses.

    The mandatory revision-label-scheme-identity argument MUST be an
    identity derived from revision-label-scheme-base.

    This extension is only valid as a top-level statement, i.e.,
    given as as a substatement to 'module' or 'submodule'. No
    substatements for this extension have been standardized.

    This extension MUST be used if there is a revision-label
    statement in the module or submodule.

    Adding a revision label scheme is a backwards-compatible version
    change. Changing a revision label scheme is a
    non-backwards-compatible version change, unless the new revision
    label scheme is backwards-compatible with the replaced revision
    label scheme. Removing a revision label scheme is a
    non-backwards-compatible version change.";

  reference
    "XXXX: Updated YANG Module Revision Handling;
    Section 3.3.1, Revision label scheme extension statement";
}

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-label.

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

    The statement MUST only be a substatement of the import
    statement. Zero, one or more 'revision-or-derived' statements
    per parent statement are allowed. No substatements for this
    extension have been standardized.
```

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.

Adding, removing or updating a 'revision-or-derived' statement to an import is a backwards-compatible change.
";

reference

"XXXX: Updated YANG Module Revision Handling;
Section 4, Import by derived revision";

}

identity revision-label-scheme-base {

description

"Base identity from which all revision label schemes are derived.";

reference

"XXXX: Updated YANG Module Revision Handling;
Section 3.3.1, Revision label scheme extension statement";

}

}

<CODE ENDS>

YANG module with augmentations to YANG Library to revision labels

<CODE BEGINS> file "ietf-yang-library-revisions@2021-06-30.yang"

module ietf-yang-library-revisions {

yang-version 1.1;

namespace

"urn:ietf:params:xml:ns:yang:ietf-yang-library-revisions";

```
prefix yl-rev;

import ietf-yang-revisions {
  prefix rev;
  reference
    "XXXX: Updated YANG Module Revision Handling";
}

import ietf-yang-library {
  prefix yanglib;
  reference "RFC 8525: YANG Library";
}

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

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

  Author:     Reshad Rahman
               <mailto:reshad@yahoo.com>

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

  Copyright (c) 2021 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.

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.";

```
// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX (including in the imports above) with
// actual RFC number and remove this note.
// RFC Ed.: please replace revision-label version with 1.0.0 and
// remove this note.
revision 2021-06-30 {
  rev:revision-label 0.2.0;
  description
    "Initial revision";
  reference
    "XXXX: 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:revision-label;
    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
      "XXXX: Updated YANG Module Revision Handling;
      Section 5.2.1, Advertising revision-label";
  }
}

augment "/yanglib:yang-library/yanglib:module-set/yanglib:module/"
  + "yanglib:submodule" {
  description
    "Augment submodule information with a revision label";
  leaf revision-label {
    type rev:revision-label;
    description
      "The revision label associated with this submodule revision.
      The label MUST match the rev:label value in the specific
```

```
        revision of the submodule included by the module loaded in
        this module-set.";

    reference
      "XXXX: Updated YANG Module Revision Handling;
       Section 5.2.1, Advertising revision-label";
  }
}

augment "/yanglib:yang-library/yanglib:module-set/"
  + "yanglib:import-only-module/yanglib:submodule" {
  description
    "Augment submodule information with a revision label";
  leaf revision-label {
    type rev:revision-label;
    description
      "The revision label associated with this submodule revision.
       The label MUST match the rev:label value in the specific
       revision of the submodule included by the
       import-only-module loaded in this module-set.";

    reference
      "XXXX: Updated YANG Module Revision Handling;
       Section 5.2.1, Advertising revision-label";
  }
}

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 boolean;
    description
      "If set to true, 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 or set to false,
       then the behavior is unspecified.";

    reference
      "XXXX: Updated YANG Module Revision Handling;
       Section 5.2.2, Reporting how deprecated and obsolete nodes
       are handled";
  }
}
```

```
leaf obsolete-nodes-absent {  
  type boolean;  
  description  
    "If set to true, this leaf indicates that the server does not  
    implement any status 'obsolete' nodes. If this leaf is  
    absent or set to false, then the behaviour is unspecified.";  
  
  reference  
    "XXXX: Updated YANG Module Revision Handling;  
    Section 5.2.2, 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 Jan Lindblad
- 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 Bo Wu

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 to thank both Anees Shaikh and Rob Shakir for their input into this problem space.

We would also like to thank Lou Berger, Andy Bierman, Martin Bjorklund, Italo Busi, Tom Hill, Scott Mansfield, Kent Watsen for their contributions and review comments.

10. Security Considerations

The document does not define any new protocol or data model. There are no security considerations beyond those specified in [RFC7950].

11. IANA Considerations

11.1. YANG Module Registrations

This document requests IANA to registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations are requested.

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

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

The following YANG module is requested to be registred in the "IANA Module Names" [RFC6020]. Following the format in RFC 6020, the following registrations are requested:

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-yang-library-revisions module:

Name: ietf-yang-library-revisions

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

Prefix: yl-rev

Reference: [RFCXXXX]

11.2. Guidance for versioning in IANA maintained YANG modules

Note for IANA (to be removed by the RFC editor): Please check that the registries and IANA YANG modules are referenced in the appropriate way.

IANA is responsible for maintaining and versioning YANG modules that are derived from other IANA registries. For example, "iana-if-type.yang" [IfTypeYang] is derived from the "Interface Types (ifType) IANA registry" [IfTypesReg], and "iana-routing-types.yang" [RoutingTypesYang] is derived from the "Address Family Numbers" [AddrFamilyReg] and "Subsequent Address Family Identifiers (SAFI) Parameters" [SAFIReg] IANA registries.

Normally, updates to the registries cause any derived YANG modules to be updated in a backwards-compatible way, but there are some cases where the registry updates can cause non-backward-compatible updates to the derived YANG module. An example of such an update is the 2020-12-31 revision of iana-routing-types.yang [RoutingTypesDecRevision], where the enum name for two SAFI values was changed.

In all cases, IANA MUST follow the versioning guidance specified in Section 3.1, and MUST include a "rev:non-backwards-compatible" substatement to the latest revision statement whenever an IANA maintained module is updated in a non-backwards-compatible way, as described in Section 3.2.

Note: For published IANA maintained YANG modules that contain non-backwards-compatible changes between revisions, a new revision should be published with the "rev:non-backwards-compatible" substatement retrospectively added to any revisions containing non-backwards-compatible changes.

Non-normative examples of updates to enumeration types in IANA maintained modules that would be classified as non-backwards-compatible changes are: Changing the status of an enumeration typedef to obsolete, changing the status of an enum entry to obsolete,

removing an enum entry, changing the identifier of an enum entry, or changing the described meaning of an enum entry.

Non-normative examples of updates to enumeration types in IANA maintained modules that would be classified as backwards-compatible changes are: Adding a new enum entry to the end of the enumeration, changing the status of an enum entry to deprecated, or improving the description of an enumeration that does not change its defined meaning.

Non-normative examples of updates to identity types in IANA maintained modules that would be classified as non-backwards-compatible changes are: Changing the status of an identity to obsolete, removing an identity, renaming an identity, or changing the described meaning of an identity.

Non-normative examples of updates to identity types in IANA maintained modules that would be classified as backwards-compatible changes are: Adding a new identity, changing the status of an identity to deprecated, or improving the description of an identity that does not change its defined meaning.

12. References

12.1. Normative References

- [I-D.ietf-netmod-rfc6991-bis]
Schoenwaelder, J., "Common YANG Data Types", draft-ietf-netmod-rfc6991-bis-06 (work in progress), April 2021.
- [I-D.ietf-netmod-yang-semver]
Claise, B., Clarke, J., Rahman, R., Wilton, R., Lengyel, B., Sterne, J., and K. D'Souza, "YANG Semantic Versioning", draft-ietf-netmod-yang-semver-02 (work in progress), February 2021.
- [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>>.

- [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>>.
- [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

- [AddrFamilyReg] "Address Family Numbers IANA Registry", <<https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>>.
- [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-13 (work in progress), March 2021.
- [I-D.ietf-netmod-yang-packages] Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", draft-ietf-netmod-yang-packages-01 (work in progress), November 2020.

- [I-D.ietf-netmod-yang-solutions]
Wilton, R., "YANG Versioning Solution Overview", draft-ietf-netmod-yang-solutions-01 (work in progress), November 2020.
- [I-D.ietf-netmod-yang-ver-selection]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Schema Selection", draft-ietf-netmod-yang-ver-selection-00 (work in progress), March 2020.
- [I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-ietf-netmod-yang-versioning-reqs-04 (work in progress), January 2021.
- [IfTypesReg]
"Interface Types (ifType) IANA Registry",
<<https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>>.
- [IfTypeYang]
"iana-if-type YANG Module",
<<https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>>.
- [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>>.
- [RoutingTypesDecRevision]
"2020-12-31 revision of iana-routing-types.yang",
<<https://www.iana.org/assignments/yang-parameters/iana-routing-types@2020-12-31.yang>>.
- [RoutingTypesYang]
"iana-routing-types YANG Module",
<<https://www.iana.org/assignments/iana-routing-types/iana-routing-types.xhtml>>.
- [SAFIReg] "Subsequent Address Family Identifiers (SAFI) Parameters IANA Registry", <<https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml>>.
- [semver] "Semantic Versioning 2.0.0", <<https://www.semver.org>>.

Appendix A. Examples of changes that are NBC

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.

Appendix B. Examples of applying the NBC change guidelines

The following sections give guidance for how some of these NBC changes could be made to a YANG module. The examples are all for "config true" nodes.

B.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 "description" updated, this is an NBC change.

If the server can support NBC revisions of the YANG module simultaneously using version selection [I-D.ietf-netmod-yang-ver-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 "description" updated, this is an NBC change.
2. Clients which require the data node select the YANG package containing the schema version they use.

B.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 when 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 then the configuration would be incomplete, this is outside the scope of this document.
4. When node "vpn-id" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

If the server can support NBC revisions of the YANG module simultaneously using version selection [I-D.ietf-netmod-yang-ver-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 YANG package containing the schema version they use
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).

B.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 B.2

B.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 when 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 "description" is updated. This is an NBC change.

If the server can support NBC revisions of the YANG module simultaneously using version selection [I-D.ietf-netmod-yang-ver-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

B.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 when 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 "description" is updated. This is an NBC change.

If the server can support NBC revisions of the YANG module simultaneously using version selection [I-D.ietf-netmod-yang-ver-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

B.6. Changing a default value

Appendix C. Changes between revisions

Note to RFC Editor (To be removed by RFC Editor)

v00 - v01

- o Removed status-description
- o Allowed both revision-date and revision-label in the filename.
- o New extension revision-label-scheme
- o To include submodules, inclusion by revision-date changed from MUST to SHOULD
- o Submodules can use revision label scheme and it can be same or different as the including module's scheme
- o Addressed various comments provided at WG adoption on rev-00

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Reshad Rahman (editor)

Email: reshad@yahoo.com

Balazs Lengyel (editor)
Ericsson

Email: balazs.lengyel@ericsson.com

Joe Clarke
Cisco Systems, Inc.

Email: jclarke@cisco.com

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

R. Wilton, Ed.
R. Rahman
J. Clarke
Cisco Systems, Inc.
J. Sterne
Nokia
B. Wu, Ed.
Huawei
November 2, 2020

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

Abstract

This document defines YANG packages, a versioned organizational structure holding a set of related YANG modules that collectively define a YANG schema. It describes how packages: are represented on a server, can be defined in offline YANG instance data files, and can be used to define the schema associated with YANG instance data files.

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 May 6, 2021.

Copyright Notice

Copyright (c) 2020 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	3
2. Introduction	4
3. Background on YANG packages	4
4. Objectives	5
5. YANG Package Definition	6
5.1. Package definition rules	7
5.2. Package versioning	8
5.2.1. Updating a package with a new version	8
5.2.1.1. Non-Backwards-compatible changes	8
5.2.1.2. Backwards-compatible changes	9
5.2.1.3. Editorial changes	9
5.2.2. YANG Semantic Versioning for packages	9
5.2.3. Revision history	10
5.3. Package conformance	10
5.3.1. Use of YANG semantic versioning	10
5.3.2. Package checksums	11
5.3.3. The relationship between packages and datastores	12
5.4. Schema referential completeness	13
5.5. Package name scoping and uniqueness	14
5.5.1. Globally scoped packages	14
5.5.2. Server scoped packages	14
5.6. Submodules packages considerations	14
5.7. Package tags	14
5.8. YANG Package Usage Guidance	15
5.8.1. Use of deviations in YANG packages	15
5.8.2. Use of features in YANG modules and YANG packages	16
5.9. YANG package core definition	16
6. Package Instance Data Files	17
7. Package Definitions on a Server	18
7.1. Package List	18
7.2. Tree diagram	19
8. YANG Library Package Bindings	19
9. YANG packages as schema for YANG instance data document	20
10. YANG Modules	20
11. Security Considerations	41
12. IANA Considerations	42
13. Open Questions/Issues	44
14. Acknowledgements	44
15. References	44

15.1. Normative References	44
15.2. Informative References	46
Appendix A. Examples	46
A.1. Example IETF Network Device YANG package	47
A.2. Example IETF Basic Routing YANG package	49
A.3. Package import conflict resolution example	52
Appendix B. Possible alternative solutions	55
B.1. Using module tags	55
B.2. Using YANG library	56
Authors' Addresses	56

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.ietf-netmod-yang-versioning-reqs].

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

- o datastore schema

This document also makes of the following terminology introduced in the YANG 1.1 Data Modeling Language [RFC7950]:

- o data node

In addition, this document defines the following terminology:

- o YANG schema: A datastore schema, not bound to any particular datastore.
- o YANG package: An organizational structure containing a collection of YANG modules, normally defined in a YANG instance data file. A YANG package defines a YANG schema by specifying a set of YANG modules and their revisions, other packages and their revisions, mandatory features, and deviations. YANG packages are defined in Section 5.
- o backwards-compatible (BC) change: When used in the context of a YANG module, it follows the definition in Section 3.1.1 of [I-D.ietf-netmod-yang-module-versioning]. When used in the

context of a YANG package, it follows the definition in Section 5.2.1.2.

- o non-backwards-compatible (NBC) change: When used in the context of a YANG module, it follows the definition in Section 3.1.2 of [I-D.ietf-netmod-yang-module-versioning]. When used in the context of a YANG package, it follows the definition in Section 5.2.1.2.
- o editorial change: When used in the context of a YANG module, it follows the definition of an 'editorial change' in 3.2 of [I-D.ietf-netmod-yang-module-versioning]. When used in the context of a YANG package, it follows the definition in Section 5.2.1.3.

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 YANG modules together into a consistent versioned definition. For example, a YANG package could define the set of YANG modules required to implement an L2VPN service on a network device. YANG packages can themselves refer to, and reuse, other package definitions.

Non-normative examples of YANG packages are provided in the appendices.

3. Background on YANG packages

It has long been acknowledged within the YANG 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 packages.

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 [I-D.ietf-netmod-yang-solutions]

to the YANG versioning requirements, [I-D.ietf-netmod-yang-versioning-reqs]. Solutions to the versioning requirements can be split into several distinct areas. [I-D.ietf-netmod-yang-module-versioning] is focused on YANG versioning scoped to individual modules. The overall solution must also consider YANG versioning and conformance scoped to YANG schema. YANG packages provide part of the solution for versioning YANG schema.

4. Objectives

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

- o To provide an alternative, simplified, YANG conformance mechanism. Rather than conformance being performed against a set of individual YANG module revisions, features, and deviations, conformance can be more simply stated in terms of YANG packages, with a set of modifications (e.g. additional modules, deviations, or features).
- o To allow YANG schema to be specified in a concise way rather than having each server explicitly list all modules, revisions, and features. YANG package definitions can be defined in documents that are available offline, and accessible via 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 define a mainly linear versioned history of sets of modules versions that are known to work together. I.e. to help mitigate the problem where 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, finding a consistent lowest common denominator set of YANG module versions may be difficult, if not impossible.

Protocol mechanisms of how clients can negotiate which packages or package versions are to be used for NETCONF/RESTCONF communications are outside the scope of this document, and are defined in [I-D.ietf-netmod-yang-ver-selection].

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 also be used to extend the functionality supported by YANG packages, if required.

5. YANG Package Definition

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

A YANG package is a versioned organizational structure defining a set of related YANG modules, packages, features, and deviations. A YANG package collectively defines a YANG schema.

Each YANG package has a name that SHOULD end with the suffix "-pkg". Package names are normally expected to be globally unique, but in some cases the package name may be locally scoped to a server or device, as described in Section 5.5.

YANG packages are versioned using the same approaches described in [I-D.ietf-netmod-yang-module-versioning] and [I-D.ietf-netmod-yang-semver]. This is described in further detail in Section 5.2.

Each YANG package version, defines:

- o some metadata about the package, e.g., description, tags, scoping, referential completeness, location information.
- o a set of YANG modules, at particular revisions, that are implemented by servers that implement the package. The modules may contain deviations.
- o a set of import-only YANG modules, at particular revisions, that are used 'import-only' by the servers that implement the package.
- o a set of included YANG packages, at particular revisions, that are also implemented by servers that implement the package.
- o a set of YANG module features that must be supported by servers that implement the package.

The structure for YANG package definitions uses existing YANG language statements, YANG Data Structure Extensions [I-D.ietf-netmod-yang-data-ext], and YANG Instance Data File Format [I-D.ietf-netmod-yang-instance-file-format].

YANG package definitions are available offline in YANG instance data files. Client applications can be designed to support particular package versions that they expect to interoperate with.

YANG package definitions are available from the server via augmentations to YANG Library [RFC8525]. Rather than client applications downloading the entire contents of YANG library to confirm that the server schema is compatible with the client, they can check, or download, a much shorter YANG package definition, and validate that it conforms to the expected schema.

YANG package definitions can also be used to define the schema associated with YANG instance data files holding other, e.g., non packages related, instance data.

5.1. Package definition rules

Packages are defined using the following rules:

1. A YANG package MAY represent a complete YANG schema or only part of a YANG schema with some module import dependencies missing, as described in Section 5.4.
2. Packages definitions are hierarchical. A package can include other packages. Only a single version of a package can be included, and conflicting package includes (e.g. from descendant package includes) MUST be explicitly resolved by indicating which version takes precedence, and which versions are being replaced.
3. For each module implemented by a package, only a single revision of that module MUST be implemented. Multiple revisions of a module MAY be listed as import-only dependencies.
4. The revision of a module listed in the package 'module' list supersedes any 'implemented' revision of the module listed in an included package module list. The 'replaces-revision' leaf-list is used to indicate which 'implemented' or 'import-only' module revisions are replaced by this module revision. This allows a package to explicitly resolve conflicts between implemented module revisions in included packages.
5. The 'replaces-revision' leaf-list in the 'import-only-module' list can be used to exclude duplicate revisions of import-only modules from included packages. Otherwise, the import-only-modules for a package are the import-only-modules from all included packages combined with any modules listed in the packages import-only-module list.

6. YANG packages definitions MAY include modules containing deviation statements, but those deviation statements MUST only be used in an RFC 7950 compatible way to indicate where a server, or class of servers, deviates from a published standard. Deviations MUST NOT be included in a package definition that is part of a published standard. See section 5.8.1 for further guidance on the use of deviations in YANG packages.

5.2. Package versioning

Individual versions of a YANG package are versioned using the "revision-label" scheme defined in section 3.3 of [I-D.ietf-netmod-yang-module-versioning].

5.2.1. Updating a package with a new version

Package compatibility is fundamentally defined by how the YANG schema between two package versions has changed.

When a package definition is updated, the version associated with the package MUST be updated appropriately, taking into consideration the scope of the changes as defined by the rules below.

A package definition SHOULD define the previous version of the package in the 'previous-version' leaf unless it is the initial version of the package. If the 'previous-version' leaf is provided then the package definition MUST set the 'nbc-changes' leaf if the new version is non-backwards-compatible with respect to the package version defined in the 'previous-version' leaf.

5.2.1.1. Non-Backwards-compatible changes

The following changes classify as non-backwards-compatible changes to a package definition:

- o Changing an 'included-package' list entry to select a package version that is non-backwards-compatible to the prior package version, or removing a previously included package.
- o Changing a 'module' or 'import-only-module' list entry to select a module revision that is non-backwards-compatible to the prior module revision, or removing a previously implemented module.
- o Removing a feature from the 'mandatory-feature' leaf-list.
- o Adding, changing, or removing a deviation that is considered a non-backwards-compatible change to the affected data node in the schema associated with the prior package version.

5.2.1.2. Backwards-compatible changes

The following changes classify as backwards-compatible changes to a package definition:

- o Changing an 'included-package' list entry to select a package version that is backwards-compatible to the prior package version, or including a new package that does not conflict with any existing included package or module.
- o Changing a 'module' or 'import-only-module' list entry to select a module revision that is backwards-compatible to the prior module revision, or including a new module to the package definition.
- o Adding a feature to the 'mandatory-feature' leaf-list.
- o Adding, changing, or removing a deviation that is considered a backwards-compatible change to the affected data node in the schema associated with the prior package version.

5.2.1.3. Editorial changes

The following changes classify as editorial changes to a package definition:

- o Changing a 'included-package' list entry to select a package version that is classified as an editorial change relative to the prior package version.
- o Changing a 'module' or 'import-only-module' list entry to select a module revision that is classified as an editorial change relative to the prior module revision.
- o Any change to any metadata associated with a package definition that causes it to have a different checksum value.

5.2.2. YANG Semantic Versioning for packages

YANG Semantic Versioning [I-D.ietf-netmod-yang-semver] MAY be used as an appropriate type of revision-label for the package version leaf.

If the format of the leaf matches the 'yangver:version' type specified in ietf-yang-semver.yang, then the package version leaf MUST be interpreted as a YANG semantic version number.

For YANG packages defined by the IETF, YANG semantic version numbers MUST be used as the version scheme for YANG packages.

The rules for incrementing the YANG package version number are equivalent to the semantic versioning rules used to version individual YANG modules, defined in section 3.2 of [I-D.ietf-netmod-yang-semver], but use the rules defined previously in Section 5.2.1 to determine whether a change is classified as non-backwards-compatible, backwards-compatible, or editorial. Where available, the semantic version number of the referenced elements in the package (included packages or modules) can be used to help determine the scope of changes being made.

5.2.3. Revision history

YANG packages do not contain a revision history. This is because packages may have many revisions and a long revision history would bloat the package definition. By recursively examining the 'previous-version' leaf of a package definition, a full revision history (including where non-backwards-compatible changes have occurred) can be dynamically constructed, if all package versions are available.

5.3. Package conformance

YANG packages allows for conformance to be checked at a package level rather than requiring a client to download all modules, revisions, and deviations from the server to ensure that the datastore schema used by the server is compatible with the client.

YANG package conformance is analogous to how YANG [RFC7950] requires that servers either implement a module faithfully, or otherwise use deviations to indicate areas of non-conformance.

For a top level package representing a datastore schema, servers **MUST** implement the package definition faithfully, including all mandatory features.

Package definitions **MAY** modify the schema for directly or hierarchically included packages through the use of different module revisions or module deviations. If the schema of any included package is modified in a non-backwards-compatible way then it **MUST** be indicated by setting the 'nbc-modified' leaf to true.

5.3.1. Use of YANG semantic versioning

Using the YANG semantic versioning scheme for package version numbers and module revision labels can help with conformance. In the general case, clients should be able to determine the nature of changes between two package versions by comparing the version number.

This usually means that a client does not have to be restricted to working only with servers that advertise exactly the same version of a package in YANG library. Instead, reasonable clients should be able to interoperate with any server that supports a package version that is backwards compatible to version that the client is designed for, assuming that the client is designed to ignore operational values for unknown data nodes.

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 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 non-backwards-compatible patch level changes.

5.3.2. Package checksums

Each YANG package definition may have a checksum associated with it to allow a client to validate that the package definition of the server matches the expected package definition without downloading the full package definition from the server.

The checksum for a package is calculated using the SHA-256 hash (XXX, reference) of the full file contents of the YANG package instance data file. This means that the checksum includes all whitespace and formatting, encoding, and all meta-data fields associated with the package and the instance data file).

The checksum for a module is calculated using the SHA-256 hash of the YANG module file definition. This means that the checksum includes all whitespace, formatting, and comments within the YANG module.

Packages that are locally scoped to a server may not have an offline instance data file available, and hence MAY not have a checksum.

The package definition allows URLs and checksums to be specified for all included packages, modules and submodules within the package definition. Checksums SHOULD be included in package definitions to validate the full integrity of the package.

On a server, package checksums SHOULD also be provided for the top level packages associated with the datastore schema.

5.3.3. The relationship between packages and datastores

As defined by NMDA [RFC8342], each datastore has an associated datastore schema. Sections 5.1 and 5.3 of NMDA defines further constraints on the schema associated with datastores. These constraints can be summarized thus:

- o The schema for all conventional datastores is the same.
- o The schema for non conventional configuration datastores (e.g., dynamic datastores) may completely differ (i.e. no overlap at all) from the schema associated with the conventional configuration datastores, or may partially or fully overlap with the schema of the conventional configuration datastores. A dynamic datastore, for example, may support different modules than conventional datastores, or may support a subset or superset of modules, features, or data nodes supported in the conventional configuration datastores. Where a data node exists in multiple datastore schema it has the same type, properties and semantics.
- o The schema for the operational datastore is intended to be a superset of all the configuration datastores (i.e. includes all the schema nodes from the conventional configuration datastores), but data nodes can be omitted if they cannot be accurately reported. The operational datastore schema can include additional modules containing only config false data nodes, but there is no harm in including those modules in the configuration datastore schema as well.

Given that YANG packages represent a YANG schema, it follows that each datastore schema can be represented using packages. In addition, the schema for most datastores on a server are often closely related. Given that there are many ways that a datastore schema could be represented using packages, the following guidance provides a consistent approach to help clients understand the relationship between the different datastore schema supported by a device (e.g., which parts of the schema are common and which parts have differences):

- o Any datastores (e.g., conventional configuration datastores) that have exactly the same datastore schema MUST use the same package definitions. This is to avoid, for example, the creation of a 'running-cfg' package and a separate 'intended-cfg' package that have identical schema.
- o Common package definitions SHOULD be used for those parts of the datastore schema that are common between datastores, when those datastores do not share exactly the same datastore schema. E.g.,

if a substantial part of the schema is common between the conventional, dynamic, and operational datastores then a single common package can be used to describe the common parts, along with other packages to describe the unique parts of each datastore schema.

- o YANG modules that do not contain any configuration data nodes SHOULD be included in the package for configuration datastores if that helps unify the package definitions.
- o The packages for the operational datastore schema MUST include all packages for all configuration datastores, along with any required modules defining deviations to mark unsupported data nodes. The deviations MAY be defined directly in the packages defining the operational datastore schema, or in separate non referentially complete packages.
- o The schema for a datastore MAY be represented using a single package or as the union of a set of compatible packages, i.e., equivalently to a set of non-conflicting packages being included together in an overarching package definition.

5.4. Schema referential completeness

A YANG package may represent a schema that is 'referentially complete', or 'referentially incomplete', indicated in the package definition by the 'complete' flag.

If all import statements in all YANG modules included in the package (either directly, or through included 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, along with locally scoped packages, to represent an update to a device's datastore schema as part of an optional software hot fix. E.g., the base software is made available as a complete globally scoped package. The hot fix is made available as an incomplete globally scoped package. A device's datastore schema can define a local package that implements the base software package updated with the hot fix package.

Referentially incomplete packages could also be used to group sets of logically related modules together, but without requiring a fixed dependency on all imported 'types' modules (e.g., iana-if-types.yang), instead leaving the choice of specific revisions of 'types' modules to be resolved when the package definition is used.

5.5. Package name scoping and uniqueness

YANG package names can be globally unique, or locally scoped to a particular server or device.

5.5.1. Globally scoped packages

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

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

5.5.2. Server scoped packages

Package definitions may be scoped to a particular server by setting the 'is-local' leaf to true in the package definition.

Locally scoped packages MAY have a package name that is not globally unique.

Locally scoped packages MAY have a definition that is not available offline from the server in a YANG instance data file.

5.6. Submodules packages considerations

As defined in [RFC7950] and [I-D.ietf-netmod-yang-semver], 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 file definition.

5.7. Package tags

[I-D.ietf-netmod-module-tags] defines YANG module tags as a mechanism to annotate a module definition with additional metadata. Tags MAY also be associated to a package definition via the 'tags' leaf-list.

The tags use the same registry and definitions used by YANG module tags.

5.8. YANG Package Usage Guidance

It is RECOMMENDED that organizations that publish YANG modules also publish YANG package definition that group and version those modules into units of related functionality. This increases interoperability, by encouraging implementations to use the same collections of YANG modules versions. Using packages also makes it easier to understand relationship between modules, and enables functionality to be described on a more abstract level than individual modules.

5.8.1. Use of deviations in YANG packages

[RFC7950] section 5.6.3 defines deviations as the mechanism to allow servers to indicate where they do not conform to a published YANG module that is being implemented.

In cases where implementations contain deviations from published packages, then those implementations SHOULD define a package that includes both the published packages and all modules containing deviations. This implementation specific package accurately reflects the schema used by the device and allows clients to determine how the implementation differs from the published package schema in an offline consumable way, e.g., when published in an instance data file (see section 6).

Organizations may wish to reuse YANG modules and YANG packages published by other organizations for new functionality. Sometimes, they may desire to modify the published YANG modules. However, they MUST NOT use deviations in an attempt to achieve this because such deviations cause two problems:

They prevent implementations from reporting their own deviations for the same nodes.

They fracture the ecosystem by preventing implementations from conforming to the standards specified by both organizations. This hurts the interoperability in the YANG community, promotes development of disconnected functional silos, and hurts creativity in the market.

5.8.2. Use of features in YANG modules and YANG packages

The YANG language supports feature statements as the mechanism to make parts of a schema optional. Published standard YANG modules SHOULD make use of appropriate feature statements to provide flexibility in how YANG modules may be used by implementations and used by YANG modules published by other organizations.

YANG packages support 'mandatory features' which allow a package to specify features that MUST be implemented by any conformant implementation of the package as a mechanism to simplify and manage the schema represented by a YANG package.

5.9. YANG package core definition

The `ietf-yang-package-types.yang` module defines a grouping to specify the core elements of the YANG package structure that is used within YANG package instance data files (`ietf-yang-package-instance.yang`) and also on the server (`ietf-yang-packages.yang`).

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

```
module: ietf-yang-package-types
```

```
  grouping yang-pkg-identification-leafs
```

```
    +-- name      pkg-name
    +-- version    pkg-version
```

```
  grouping yang-pkg-instance
```

```
    +-- name                pkg-name
    +-- version              pkg-version
    +-- timestamp?          yang:date-and-time
    +-- organization?       string
    +-- contact?            string
    +-- description?        string
    +-- reference?          string
    +-- complete?           boolean
    +-- local?              boolean
    +-- previous-version?   pkg-version
    +-- nbc-changes?        boolean
    +-- tag*                tags:tag
    +-- mandatory-feature*  scoped-feature
    +-- included-package* [name version]
      | +-- name                pkg-name
      | +-- version              pkg-version
      | +-- replaces-version*   pkg-version
```

```

    +-- nbc-modified?          boolean
    +-- location*              inet:uri
    +-- checksum?              pkg-types:sha-256-hash
+-- module* [name]
    +-- name                    yang:yang-identifier
    +-- revision?              rev:revision-date-or-label
    +-- replaces-revision*     rev:revision-date-or-label
    +-- namespace?             inet:uri
    +-- location*              inet:uri
    +-- checksum?              pkg-types:sha-256-hash
    +-- submodule* [name]
        +-- name?              yang:yang-identifier
        +-- revision            yang:revision-identifier
        +-- location*           inet:uri
        +-- checksum?           pkg-types:sha-256-hash
+-- import-only-module* [name revision]
    +-- name?                  yang:yang-identifier
    +-- revision?              rev:revision-date-or-label
    +-- replaces-revision*     rev:revision-date-or-label
    +-- namespace?             inet:uri
    +-- location*              inet:uri
    +-- checksum?              pkg-types:sha-256-hash
    +-- submodule* [name]
        +-- name?              yang:yang-identifier
        +-- revision            yang:revision-identifier
        +-- location*           inet:uri
        +-- checksum?           pkg-types:sha-256-hash

```

6. Package Instance Data Files

YANG packages SHOULD be available offline from the server, defined as YANG instance data files [I-D.ietf-netmod-yang-instance-file-format] using the YANG schema below to define the package data.

The following rules apply to the format of the YANG package instance files:

1. The file SHOULD be encoded in JSON.
2. The name of the file SHOULD follow the format "<package-name>@<version>.json".
3. The package name MUST be specified in both the instance-data-set 'name' and package 'name' leafs.
4. The 'description' field of the instance-data-set SHOULD be "YANG package definition".

5. The 'timestamp', 'organization', 'contact' fields are defined in both the instance-data-set metadata and the YANG package metadata. Package definitions SHOULD only define these fields as part of the package definition. If any of these fields are populated in the instance-data-set metadata then they MUST contain the same value as the corresponding leaves in the package definition.
6. The 'revision' list in the instance data file SHOULD NOT be used, since versioning is handled by the package definition.
7. The instance data file 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
```

```
    structure package:
      // Uses the yang-package-instance grouping defined in
      // ietf-yang-package-types.yang
      +-- name                pkg-name
      +-- version             pkg-version
      ... remainder of yang-package-instance grouping ...
```

7. Package Definitions on a Server

7.1. Package List

A top level 'packages' container holds the list of all versions of all packages known to the server. Each list entry uses the common package definition, but with the addition of package location and checksum information that cannot be contained within a offline package definition contained in an instance data file.

The '/packages/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.

7.2. Tree diagram

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

```
module: ietf-yang-packages
  +--ro packages
    +--ro package* [name version]
      // Uses the yang-package-instance grouping defined in
      // ietf-yang-package-types.yang, with location and checksum:
      +--ro name                pkg-name
      +--ro version              pkg-version
      ... remainder of yang-package-instance grouping ...
      +--ro location*            inet:uri
      +--ro checksum?            pkg-types:sha-256-hash
```

8. YANG Library Package Bindings

The YANG packages module also augments YANG library to allow a server to optionally indicate that a datastore schema is defined by a package, or a union of compatible packages. Since packages can generally be made available offline in instance data files, it may be sufficient for a client to only check that a compatible version of the package is implemented by the server without fetching either the package definition, or downloading and comparing the full list of modules and enabled features.

If a server indicates that a datastore schema maps to a particular package, then it **MUST** exactly match the schema defined by that package, taking into account enabled features and any deviations.

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 included package, and the new package can define additional modules containing deviations to the modules in the original package, allowing the new package to accurately describe the server's behavior. There is no specific mechanism provided to indicate that a mandatory-feature in package definition is not supported on a server, but deviations **MAY** be used to disable functionality predicated by an if-feature statement.

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

```
module: ietf-yl-packages
  augment /yanglib:yang-library/yanglib:schema:
    +--ro package* [name version]
      +--ro name      -> /pkgs:packages/package/name
      +--ro version    leafref
      +--ro checksum?  leafref
```

9. YANG packages as schema for YANG instance data document

YANG package definitions can be used as the schema definition for YANG instance data files. When using a package schema, the name and version of the package MUST be specified, a package checksum and/or URL to the package definition MAY also be provided.

The "ietf-yang-inst-data-pkg" YANG module has the following structure:

```
module: ietf-yang-inst-data-pkg

  augment-structure /yid:instance-data-set/yid:content-schema-spec:
    +--:(pkg-schema)
      +-- pkg-schema
        +-- name      pkg-name
        +-- version    pkg-version
        +-- location*  inet:uri
        +-- checksum?  pkg-types:sha-256-hash
```

10. YANG Modules

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

```
<CODE BEGINS> file "ietf-yang-package-types@2020-01-21.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-revisions {
    prefix rev;
    reference "XXXX: Updated YANG Module Revision Handling";
```

```
}

import ietf-yang-types {
  prefix yang;
  rev:revision-or-derived 2019-07-21;
  reference "RFC 6991bis: Common YANG Data Types.";
}

import ietf-inet-types {
  prefix inet;
  rev:revision-or-derived 2013-07-15;
  reference "RFC 6991: Common YANG Data Types.";
}

import ietf-module-tags {
  prefix tags;
  // RFC Ed. Fix revision once revision date of
  // ietf-module-tags.yang is known.
  reference "RFC XXX: 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) 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.
```

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.";

```
// 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 2020-01-21 {
  rev:revision-label 0.2.0;
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}
```

```
/*
 * Typedefs
 */
```

```
typedef pkg-name {
  type yang:yang-identifier;
  description
    "Package names are typed as YANG identifiers.";
}
```

```
typedef pkg-version {
  type rev:revision-date-or-label;
  description
    "Package versions SHOULD be a revision-label (e.g. perhaps a
    YANG Semver version string). Package versions MAY also be a
    revision-date";
}
```

```
typedef pkg-identifier {
  type rev:name-revision;
  description
    "Package identifiers combine a pkg-name and a pkg-version";
}
```

```
typedef scoped-feature {
  type string {
    pattern '[a-zA-Z_][a-zA-Z0-9\-\_\.]*:[a-zA-Z_][a-zA-Z0-9\-\_\.]*';
  }
}
```

```
description
  "Represents a feature name scoped to a particular module,
   identified as the '<module-name>:<feature-name>', where both
   <module-name> and <feature-name> are YANG identifier strings,
   as defiend by Section 12 or RFC 6020.";
reference
  "RFC XXXX, YANG Packages.";
}

typedef sha-256-hash {
  type string {
    length "64";
    pattern "[0-9a-fA-F]*";
  }
  description
    "A SHA-256 hash represented as a hexadecimal string.

    Used as the checksum for modules, submodules and packages in a
    YANG package definition.

    For modules and submodules the SHA-256 hash is calculated on
    the contents of the YANG file defining the module/submodule.

    For packages the SHA-256 hash is calculated on the file
    containing the YANG instance data document holding the package
    definition";
}

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

  leaf name {
    type pkg-name;
    mandatory true;
    description
      "The YANG package name.";
  }

  leaf version {
    type pkg-version;
    mandatory true;
    description
```

```
        "Uniquely identifies a particular version of a YANG package.

        Follows the definition for revision labels defined in
        draft-verdt-nemod-yang-module-versioning, section XXX";
    }
}

grouping yang-pkg-instance {
    description
        "Specifies the data node for a full YANG package instance
        represented either on a server or as a YANG instance data
        document.";
    uses yang-pkg-identification-leafs;

    leaf timestamp {
        type yang:date-and-time;

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

    leaf organization {
        type string;

        description "Organization responsible for this package";
    }

    leaf contact {
        type string;

        description
            "Contact information for the person or organization to whom
            queries concerning this package should be sent.";
    }

    leaf description {
        type string;

        description "Provides a description of the package";
    }

    leaf reference {
        type string;

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

```
leaf 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 included packages.";
}

leaf local {
  type boolean;
  default false;
  description
    "Defines that the package definition is local to the server,
    and the name of the package MAY not be unique, and the
    package definition MAY not be available in an offline file.

    Local packages can be used when the schema for the device
    can be changed at runtime through the addition or removal of
    software packages, or hot fixes.";
}

leaf previous-version {
  type pkg-version;
  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 nbc-changes {
  type boolean;
  default false;
  description
    "Indicates whether the defined package version contains
    non-backwards-compatible changes relative to the package
    version defined in the 'previous-version' leaf.";
}

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-list mandatory-feature {
  type scoped-feature;
  description
    "Lists features from any modules included in the package that
    MUST be supported by any server implementing the package.

    Features already specified in a 'mandatory-feature' list of
    any included package MUST also be supported by server
    implementations and do not need to be repeated in this list.

    All other features defined in modules included in the
    package are OPTIONAL to implement.

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

list included-package {
  key "name version";
  description
    "An entry in this list represents a package that is included
    as part of the package definition, or an indirectly included
    package that is changed in a non backwards compatible way.

    It can be used to resolve inclusion of conflicting package
    versions by explicitly specifying which package version is
    used.

    If included 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.

    If the schema for any packages that are included, either
    directly or indirectly via another package include, are
    changed in any non-backwards-compatible way then they MUST
    be explicitly listed in the included-packages list with the
    'nbc-modified' leaf set to true.

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

  uses yang-pkg-identification-leafs;
```

```
leaf-list replaces-version {
  type pkg-version;
  description
    "Gives the version of an included package version that
     is replaced by this included package revision.";
}

leaf nbc-modified {
  type boolean;
  default false;
  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 incompatible revision. This could also
     occur if a module's revision was replaced by an earlier
     revision that had the effect of removing some data
     nodes.";
}

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 checksum {
  type pkg-types:sha-256-hash;
  description
    "The SHA-256 hash calculated on the textual package
     definition, represented as a hexadecimal string.";
}

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 revision 'implemented' by an included package. Any replaced module revision SHOULD also be listed in the 'replaces-revision' list.";

reference

"RFC 7950: The YANG 1.1 Data Modeling Language.";

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

```
leaf revision {  
  type rev:revision-date-or-label;  
  description  
    "The YANG module revision date or revision-label.
```

If no revision statement is present in the YANG module,
this leaf is not instantiated.";

```
}
```

```
leaf-list replaces-revision {  
  type rev:revision-date-or-label;  
  description  
    "Gives the revision of an module (implemented or  
    import-only) defined in an included package that is  
    replaced by this implemented module revision.";  
}
```

```
leaf namespace {  
  type inet:uri;  
  description  
    "The XML namespace identifier for this module.";  
}
```

```
leaf-list location {  
  type inet:uri;  
  description  
    "Contains a URL that represents the YANG schema resource  
    for this module.  
  
    This leaf will only be present if there is a URL available  
    for retrieval of the schema for this entry.";  
}
```

```
leaf checksum {
  type pkg-types:sha-256-hash;
  description
    "The SHA-256 hash calculated on the textual module
    definition, represented as a hexadecimal string.";
}

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 yang: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.";
  }

  leaf-list location {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema resource
      for this submodule.

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

  leaf checksum {
    type pkg-types:sha-256-hash;
    description
      "The SHA-256 hash calculated on the textual submodule
      definition, represented as a hexadecimal string.";
  }
}

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 rev:revision-date-or-label;
  description
    "The YANG module revision date or revision-label.

    If no revision statement is present in the YANG module,
    this leaf is not instantiated.";
}

leaf-list replaces-revision {
  type rev:revision-date-or-label;
  description
    "Gives the revision of an import-only-module defined in an
    included package that is replaced by this
    import-only-module revision.";
}

leaf namespace {
  type inet:uri;
  description
    "The XML namespace identifier for this module.";
}

leaf-list location {
  type inet:uri;
  description
    "Contains a URL that represents the YANG schema resource
    for this module.

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

```
    }

    leaf checksum {
      type pkg-types:sha-256-hash;
      description
        "The SHA-256 hash calculated on the textual submodule
        definition, represented as a hexadecimal string.";
    }

    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 yang: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.";
      }

      leaf-list location {
        type inet:uri;
        description
          "Contains a URL that represents the YANG schema resource
          for this submodule.

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

      leaf checksum {
        type pkg-types:sha-256-hash;
        description
          "The SHA-256 hash calculated on the textual submodule
          definition, represented as a hexadecimal string.";
      }
    }
  }
}
```

```
    }  
  }  
<CODE ENDS>
```

```
<CODE BEGINS> file "ietf-yang-package-instance@2020-01-21.yang"  
module ietf-yang-package-instance {  
  yang-version 1.1;  
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-package-instance";  
  prefix pkg-inst;  
  
  import ietf-yang-revisions {  
    prefix rev;  
    reference "XXXX: Updated YANG Module Revision Handling";  
  }  
  
  import ietf-yang-package-types {  
    prefix pkg-types;  
    rev:revision-or-derived 0.2.0;  
    reference "RFC XXX: YANG Schema Versioning.";  
  }  
  
  import ietf-yang-structure-ext {  
    prefix sx;  
    reference "RFC XXX: YANG Data Structure Extensions.";  
  }  
  
  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) 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.

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.";

```
// 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 2020-01-21 {
  rev:revision-label 0.2.0;
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}

/*
 * Top-level structure
 */

sx:structure package {
  description
    "Defines the YANG package structure for use in a YANG instance
    data document.";

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

```
<CODE BEGINS> file "ietf-yang-package@2020-01-21.yang"
module ietf-yang-packages {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-packages";
  prefix pkgs;
```

```
import ietf-yang-revisions {
  prefix rev;
  reference "XXXX: Updated YANG Module Revision Handling";
}

import ietf-yang-package-types {
  prefix pkg-types;
  rev:revision-or-derived 0.2.0;
  reference "RFC XXX: YANG Packages.";
}

import ietf-inet-types {
  prefix inet;
  rev:revision-or-derived 2013-07-15;
  reference "RFC 6991: Common YANG Data Types.";
}

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 defines YANG packages on a server implementation.

  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.

  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.";
```

```
// 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 2020-01-21 {
    rev:revision-label 0.2.0;
    description
        "Initial revision";
    reference
        "RFC XXXX: YANG Packages";
}

/*
 * Groupings
 */

grouping yang-pkg-ref {
    description
        "Defines the leaves used to reference a single YANG package";

    leaf name {
        type leafref {
            path '/pkgs:packages/pkgs:package/pkgs:name';
        }
        description
            "The name of the references package.";
    }

    leaf version {
        type leafref {
            path '/pkgs:packages'
                + '/pkgs:package[pkgs:name = current()/../name]'
                + '/pkgs:version';
        }
        description
            "The version of the referenced package.";
    }

    leaf checksum {
        type leafref {
            path '/pkgs:packages'
                + '/pkgs:package[pkgs:name = current()/../name]'
                + '[pkgs:version = current()/../version]/pkgs:checksum';
        }
        description
```

```
        "The checksum of the referenced package.";
    }
}

grouping yang-ds-pkg-ref {
    description
        "Defines the list used to reference a set of YANG packages that
        collectively represent a datastore schema.";

    list package {
        key "name version";

        description
            "Identifies the YANG packages that collectively defines the
            schema for the associated datastore.

            The datastore schema is defined as the union of all
            referenced packages, that MUST represent a referentially
            complete schema.

            All of the referenced packages must be compatible with no
            conflicting module versions or dependencies.";

        uses yang-pkg-ref;
    }
}

/*
 * Top level data nodes.
 */

container packages {
    config false;
    description "All YANG package definitions";

    list package {
        key "name version";

        description
            "YANG package instance";

        uses pkg-types:yang-pkg-instance;

        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 checksum {  
  type pkg-types:sha-256-hash;  
  description  
    "The checksum of the package this schema relates to,  
    calculated on the 'YANG instance data file' package  
    definition available in the 'location' leaf list.  
  
    This leaf MAY be omitted if the referenced package is  
    locally scoped without an associated checksum.";  
}  
}  
}  
}  
<CODE ENDS>
```

```
<CODE BEGINS> file "ietf-yl-package@2020-01-21.yang"  
module ietf-yl-packages {  
  yang-version 1.1;  
  namespace "urn:ietf:params:xml:ns:yang:ietf-yl-packages";  
  prefix yl-pkgs;  
  
  import ietf-yang-revisions {  
    prefix rev;  
    reference "XXXX: Updated YANG Module Revision Handling";  
  }  
  
  import ietf-yang-packages {  
    prefix pkgs;  
    rev:revision-or-derived 0.2.0;  
    reference "RFC XXX: YANG Packages.";  
  }  
  
  import ietf-yang-library {  
    prefix yanglib;  
    rev:revision-or-derived 2019-01-04;  
    reference "RFC 8525: 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.

  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.";

// 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 2020-01-21 {
  rev:revision-label 0.2.0;
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}
```

```
/*
 * Augmentations
 */

augment "/yanglib:yang-library/yanglib:schema" {
  description
    "Allow datastore schema to be related to a set of YANG
    packages";

  uses pkgs:yang-ds-pkg-ref;
}
}
<CODE ENDS>

<CODE BEGINS> file "ietf-yang-inst-data-pkg@2020-01-21.yang"
module ietf-yang-inst-data-pkg {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-inst-data-pkg";
  prefix yid-pkg;

  import ietf-yang-revisions {
    prefix rev;
    reference "XXXX: Updated YANG Module Revision Handling";
  }

  import ietf-yang-package-types {
    prefix pkg-types;
    rev:revision-or-derived 0.2.0;
    reference "RFC XXX: YANG Schema Versioning.";
  }

  import ietf-yang-structure-ext {
    prefix sx;
    reference "RFC XXX: YANG Data Structure Extensions.";
  }

  import ietf-yang-instance-data {
    prefix yid;
    reference "RFC XXX: YANG Instance Data File Format.";
  }

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

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

"The module augments ietf-yang-instance-data to allow package definitions to be used to define schema in YANG instance data documents.

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.

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.";

// 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 2020-01-21 {  
  rev:revision-label 0.2.0;  
  description  
    "Initial revision";  
  reference  
    "RFC XXXX: YANG Packages";  
}
```

```
/*  
 * Augmentations
```

```
*/  
  
sx:augment-structure  
  "/yid:instance-data-set/yid:content-schema-spec" {  
    description  
      "Add package reference to instance data set schema  
      specification";  
    case pkg-schema {  
      container pkg-schema {  
        uses pkg-types:yang-pkg-identification-leafs;  
  
        leaf checksum {  
          type pkg-types:sha-256-hash;  
          description  
            "The SHA-256 hash of the package, calculated on  
            the textual package definition, represented as a  
            hexadecimal string.";  
        }  
  
        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";  
        }  
      }  
    }  
  }  
}  
<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 files, 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 files to restrict access to sensitive information. SHA-256 checksums are used to ensure the integrity of YANG package definitions, imported modules, and sub-modules.

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 YANG packages 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 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.

This document requests IANA to registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations are requested.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-package-types.yang

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-package-instance.yang

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-packages.yang

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yl-packages.yang

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-inst-data-pkg.yang

Registrant Contact: The IESG.

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

This document requests that the following YANG modules are added in the "YANG Module Names" registry [RFC6020]:

Name: ietf-yang-package-types.yang

Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-package-types.yang

Prefix: pkg-types

Reference: RFC XXXX

Name: ietf-yang-package-instance.yang

Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-package-instance.yang

Prefix: pkg-inst

Reference: RFC XXXX

Name: ietf-yang-packages.yang

Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-packages.yang

Prefix: pkgs

Reference: RFC XXXX

Name: ietf-yl-packages.yang

Namespace: urn:ietf:params:xml:ns:yang:ietf-yl-packages.yang

Prefix: yl-pkgs

Reference: RFC XXXX

Name: ietf-yang-inst-data-pkg.yang

Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-inst-data-pkg.yang

Prefix: yid-pkg

Reference: RFC XXXX

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, James Cumming, Mahesh Jethanandani, Balazs Lengyel, Ladislav Lhotka, and Jan Lindblad.

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-10 (work in progress), February 2020.
- [I-D.ietf-netmod-yang-data-ext]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", draft-ietf-netmod-yang-data-ext-05 (work in progress), December 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-12 (work in progress), April 2020.
- [I-D.ietf-netmod-yang-module-versioning]
Wilton, R., Rahman, R., Lengyel, B., Clarke, J., Sterne, J., Claise, B., and K. D'Souza, "Updated YANG Module Revision Handling", draft-ietf-netmod-yang-module-versioning-01 (work in progress), July 2020.
- [I-D.ietf-netmod-yang-semver]
Claise, B., Clarke, J., Rahman, R., Wilton, R., Lengyel, B., Sterne, J., and K. D'Souza, "YANG Semantic Versioning", draft-ietf-netmod-yang-semver-01 (work in progress), July 2020.

- [I-D.ietf-netmod-yang-solutions]
Wilton, R., "YANG Versioning Solution Overview", draft-ietf-netmod-yang-solutions-00 (work in progress), March 2020.
- [I-D.ietf-netmod-yang-ver-selection]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and W. Bo, "YANG Schema Selection", draft-ietf-netmod-yang-ver-selection-00 (work in progress), March 2020.
- [I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-ietf-netmod-yang-versioning-reqs-03 (work in progress), June 2020.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [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>>.
- [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>>.

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., Auerswald, E., Farrel, A., and Q. WU, "Handling Long Lines in Inclusions in Internet-Drafts and RFCs", draft-ietf-netmod-artwork-folding-12 (work in progress), January 2020.
- [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. 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. For brevity, the examples exclude namespace declarations, and use a shortened URL of "tiny.cc/ietf-yang" as a replacement for "https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC".

A.1. Example IETF Network Device YANG package

This section provides an instance data file 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",
    "pkg-schema": {
      package: "ietf-yang-package-defn-pkg@0.1.0.json"
    },
    "description": "YANG package definition",
    "content-data": {
      "ietf-yang-package-instance:yang-package": {
        "name": "example-ietf-network-device-pkg",
        "version": "1.1.2",
        "timestamp": "2018-12-13T17:00:00Z",
        "organization": "IETF NETMOD Working Group",
        "contact" : "WG Web:  <http://tools.ietf.org/wg/netmod/>, \
                    WG List: <mailto:netmod@ietf.org>",
        "description": "Example IETF network device YANG package.\
        \
        This package defines a small sample set of \
        YANG modules that could represent the basic set of \
        modules that a standard network device might be expected \
        to support.",
      }
    }
  }
}
```

```
"reference": "XXX, draft-rwilton-netmod-yang-packages",
"location": [ "file://example.org/yang/packages/\
               ietf-network-device@v1.1.2.json" ],
"module": [
  {
    "name": "iana-crypt-hash",
    "revision": "2014-08-06",
    "location": [ "https://tiny.cc/ietf-yang/\
                  iana-crypt-hash%402014-08-06.yang" ],
    "checksum": "fa9fde408ddec2c16bf2c6b9e4c2f80b\
                813a2f9e48c127016f3fa96da346e02d"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-system%402014-08-06.yang" ],
    "checksum": "8a692ee2521b4ffe87a88303a61a1038\
                79ee26bff050c1b05a2027ae23205d3f"
  },
  {
    "name": "ietf-interfaces",
    "revision": "2018-02-20",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-interfaces%402018-02-20.yang" ],
    "checksum": "f6faea9938f0341ed48fda93dba9a69a\
                a32ee7142c463342efec3d38f4eb3621"
  },
  {
    "name": "ietf-netconf-acm",
    "revision": "2018-02-14",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-netconf-acm%402018-02-14.yang" ],
    "checksum": "e03f91317f9538a89296e99df3ff0c40\
                03cdfa70bf517407643b3ec13c1ed25"
  },
  {
    "name": "ietf-key-chain",
    "revision": "2017-06-15",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-key-chain@2017-06-15.yang" ],
    "checksum": "6250705f59fc9ad786e8d74172ce90d5\
                8deec437982cbca7922af40b3ae8107c"
  },
  {
    "name": "ietf-ip",
    "revision": "2018-02-22",
    "location": [ "https://tiny.cc/ietf-yang/"
```

```

        ietf-ip%402018-02-22.yang" ],
    "checksum": "b624c84a66c128ae69ab107a5179ca8e\
                20e693fb57dbe5cb56c3db2ebb18c894"
  },
],
"import-only-module": [
  {
    "name": "ietf-yang-types",
    "revision": "2013-07-15",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-yang-types%402013-07-15.yang" ],
    "checksum": "a04cdcc875764a76e89b7a0200c6b9d8\
                00b10713978093acda7840c7c2907c3f"
  },
  {
    "name": "ietf-inet-types",
    "revision": "2013-07-15",
    "location": [ "https://tiny.cc/ietf-yang/\
                  ietf-inet-types%402013-07-15.yang" ],
    "checksum": "12d98b0143a5ca5095b36420f9ebc1ff\
                a61cfd2eaa850080244cadf01b86ddf9"
  }
]
}
}
}
}
}
}
}
<CODE ENDS>

```

A.2. Example IETF Basic Routing YANG package

This section provides an instance data file 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: '\ ' line wrapping per BCP XX (RFC XXXX) =====

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-ietf-routing-pkg",
    "module": [ "ietf-yang-package@2019-09-11.yang" ],
    "description": "YANG package definition",
    "content-data": {
      "ietf-yang-package-instance:yang-package": {
        "name": "example-ietf-routing",
        "version": "1.3.1",
        "timestamp": "2018-12-13T17:00:00Z",
        "description": "This package defines a small sample set of \
          IETF routing YANG modules that could represent the set of \
          IETF routing functionality that a basic IP network device \
          might be expected to support.",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "imported-packages": [
          {
            "name": "ietf-network-device",
            "version": "1.1.2",
            "location": [ "http://example.org/yang/packages/\
              ietf-network-device@v1.1.2.json" ],
            "checksum": ""
          }
        ],
        "module": [
          {
            "name": "ietf-routing",
            "revision": "2018-03-13",
            "location": [ "https://tiny.cc/ietf-yang/\
              ietf-routing@2018-03-13.yang" ],
            "checksum": ""
          },
          {
            "name": "ietf-ipv4-unicast-routing",
            "revision": "2018-03-13",
            "location": [ "https://tiny.cc/ietf-yang/\
              ietf-ipv4-unicast-routing@2018-03-13.yang" ],
            "checksum": ""
          },
          {
            "name": "ietf-ipv6-unicast-routing",
            "revision": "2018-03-13",
            "location": [ "https://tiny.cc/ietf-yang/\
              ietf-ipv6-unicast-routing@2018-03-13.yang" ],
            "checksum": ""
          }
        ]
      }
    }
  }
}
```

```
    },
    {
      "name": "ietf-isis",
      "revision": "2018-12-11",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    },
    {
      "name": "ietf-interfaces-common",
      "revision": "2018-07-02",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    },
    {
      "name": "ietf-if-l3-vlan",
      "revision": "2017-10-30",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    },
    {
      "name": "ietf-routing-policy",
      "revision": "2018-10-19",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    },
    {
      "name": "ietf-bgp",
      "revision": "2018-05-09",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    },
    {
      "name": "ietf-access-control-list",
      "revision": "2018-11-06",
      "location": [ "https://tiny.cc/ietf-yang/\n" ],
      "checksum": ""
    }
  ],
  "import-only-module": [
    {
      "name": "ietf-routing-types",
      "revision": "2017-12-04",
```

```

        "location": [ "https://tiny.cc/ietf-yang/\
                      ietf-routing-types@2017-12-04.yang" ],
        "checksum": ""
    },
    {
        "name": "iana-routing-types",
        "revision": "2017-12-04",
        "location": [ "https://tiny.cc/ietf-yang/\
                      iana-routing-types@2017-12-04.yang" ],
        "checksum": ""
    },
    {
        "name": "ietf-bgp-types",
        "revision": "2018-05-09",
        "location": [ "https://tiny.cc/ietf-yang/\
                      " ],
        "checksum": ""
    },
    {
        "name": "ietf-packet-fields",
        "revision": "2018-11-06",
        "location": [ "https://tiny.cc/ietf-yang/\
                      " ],
        "checksum": ""
    },
    {
        "name": "ietf-ethertypes",
        "revision": "2018-11-06",
        "location": [ "https://tiny.cc/ietf-yang/\
                      " ],
        "checksum": ""
    }
]
}
}
}
}
<CODE ENDS>

```

A.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-instance: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-instance: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",
        "included-package": [
          {
            "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 " ]
      }
    ]
  }
}
```

Appendix B. Possible alternative solutions

This section briefly describes some alternative solutions. It can be removed if this document is adopted as a WG draft.

B.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 files 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.

B.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 file. The use of YANG packages offers 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 must 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 with secure checksums. YANG library's use of a 'content-id' is unique only to the device that generated them.
3. Packages may be versioned using a semantic versioning scheme, YANG library does not provide a schema level semantic version number.
4. For a YANG library instance data file to contain the necessary information, it probably needs both YANG library and various augmentations (e.g. to include each module's semantic version number), unless a new version of YANG library is defined containing this information. The module definition for a YANG package is specified to contain all of the necessary information to solve the problem without augmentations
5. YANG library is designed to publish information about the modules, datastores, and datastore schema used by a server. The information required to construct an off box schema is not precisely the same, and hence the definitions might deviate from each other over time.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Reshad Rahman
Cisco Systems, Inc.

Email: rrahman@cisco.com

Joe Clarke
Cisco Systems, Inc.

Email: jclarke@cisco.com

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Bo Wu (editor)
Huawei

Email: lana.wubo@huawei.com

Network Working Group
Internet-Draft
Updates: 8407 (if approved)
Intended status: Standards Track
Expires: 13 January 2022

B. Claise
Huawei
J. Clarke, Ed.
R. Rahman
R. Wilton, Ed.
Cisco Systems, Inc.
B. Lengyel
Ericsson
J. Sterne
Nokia
K. D'Souza
AT&T
12 July 2021

YANG Semantic Versioning
draft-ietf-netmod-yang-semver-03

Abstract

This document specifies a scheme and guidelines for applying a modified set of semantic versioning rules to revisions of YANG modules. Additionally, this document defines a revision-label for this modified semver scheme.

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 13 January 2022.

Copyright Notice

Copyright (c) 2021 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. Terminology and Conventions	3
3. YANG Semantic Versioning	3
3.1. YANG Semantic Versioning Pattern	3
3.2. Semantic Versioning Scheme for YANG Artifacts	4
3.2.1. Examples for YANG semantic version numbers	6
3.3. YANG Semantic Version Update Rules	8
3.4. Examples of the YANG Semver Label	10
3.4.1. Example Module Using YANG Semver	10
3.4.2. Example of Package Using YANG Semver	12
4. Import Module by Semantic Version	12
5. Guidelines for Using Semver During Module Development	13
5.1. Pre-release Version Precedence	14
5.2. YANG Semver in IETF Modules	15
6. YANG Module	16
7. Contributors	17
8. Security Considerations	18
9. IANA Considerations	18
9.1. YANG Module Registrations	18
9.2. Guidance for YANG Semver in IANA maintained YANG modules and submodules	19
10. References	19
10.1. Normative References	19
10.2. Informative References	20
Appendix A. Example IETF Module Development	21
Authors' Addresses	22

1. Introduction

[I-D.ietf-netmod-yang-module-versioning] puts forth a number of concepts relating to modified rules for updating modules and submodules, a means to signal when a new revision of a module or submodule has non-backwards-compatible (NBC) changes compared to its previous revision, and a versioning scheme that uses the revision history as a lineage for determining from where a specific revision of a YANG module or submodule is derived. Additionally, section 3.3 of [I-D.ietf-netmod-yang-module-versioning] defines a revision label

which can be used as an overlay or alias to provide additional context or an additional way to refer to a specific revision.

This document defines a revision-label scheme that uses modified [semver] rules for YANG artifacts (i.e., YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages]) as well as the revision label definition for using this scheme. The goal of this is to add a human readable version label that provides compatibility information for the YANG artifact without one needing to compare or parse its body. The label and rules defined herein represent the RECOMMENDED revision label scheme for IETF YANG artifacts.

Note that a specific revision of the Semver 2.0.0 specification is referenced here (from June 19, 2020) to provide an immutable version. This is because the 2.0.0 version of the specification has changed over time without any change to the semantic version itself. In some cases the text has changed in non-backwards-compatible ways.

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.

Additionally, this document uses the following terminology:

- * YANG artifact: YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages] , and YANG schema elements are examples of YANG artifacts for the purposes of this document.

3. YANG Semantic Versioning

This section defines YANG Semantic Versioning, explains how it is used with YANG artifacts, and the rules associated with changing an artifact's semantic version number when its contents are updated.

3.1. YANG Semantic Versioning Pattern

YANG artifacts that employ semantic versioning as defined in this document MUST use a version string (e.g., in revision-label or as a package version) that corresponds to the following pattern:
X.Y.Z_COMPAT. Where:

- * X, Y and Z are mandatory non-negative integers that are each less than 2147483647 (i.e., the maximum signed 32-bit integer value) and MUST NOT contain leading zeroes
- * The '.' is a literal period (ASCII character 0x2e)
- * The '_' is an optional single literal underscore (ASCII character 0x5f) and MUST only be present if the following COMPAT element is included
- * COMPAT, if it is specified, MUST be either the literal string "compatible" or the literal string "non_compatible"

Additionally, [semver] defines two specific types of metadata that may be appended to a semantic version string. Pre-release metadata MAY be appended to a semver string after a trailing '-' character. Build metadata MAY be appended after a trailing '+' character. If both pre-release and build metadata are present, then build metadata MUST follow pre-release metadata. While build metadata MUST be ignored by YANG semver parsers, pre-release metadata MUST be used during module and submodule development and MUST be considered base on Section 5 . Both pre-release and build metadata are allowed in order to support all of the [semver] rules. Thus, a version lineage that follows strict [semver] rules is allowed for a YANG artifact.

To signal the use of this versioning scheme, modules and submodules MUST set the revision-label-scheme extension as defined in [I-D.ietf-netmod-yang-module-versioning] to the identity "yang-semver". That identity value is defined in the ietf-yang-semver module below.

Additionally, this ietf-yang-semver module defines a typedef that formally specifies the syntax of the YANG semver version string.

3.2. Semantic Versioning Scheme for YANG Artifacts

This document defines the YANG semantic versioning scheme that is used for YANG artifacts that employ the YANG semver label. The versioning scheme has the following properties:

- * The YANG semantic versioning scheme is extended from version 2.0.0 of the semantic versioning scheme defined at semver.org [semver] to cover the additional requirements for the management of YANG artifact lifecycles that cannot be addressed using the semver.org 2.0.0 versioning scheme alone.

- * Unlike the [semver] versioning scheme, the YANG semantic versioning scheme supports updates to older versions of YANG artifacts, to allow for bug fixes and enhancements to artifact versions that are not the latest. However, it does not provide for the unlimited branching and updating of older revisions which are documented by the general rules in [I-D.ietf-netmod-yang-module-versioning] .
- * YANG artifacts that follow the [semver] versioning scheme are fully compatible with implementations that understand the YANG semantic versioning scheme defined in this document.
- * If updates are always restricted to the latest revision of the artifact only, then the version numbers used by the YANG semantic versioning scheme are exactly the same as those defined by the [semver] versioning scheme.

Every YANG module and submodule versioned using the YANG semantic versioning scheme specifies the module's or submodule's semantic version number as the argument to the 'rev:revision-label' statement.

Because the rules put forth in [I-D.ietf-netmod-yang-module-versioning] are designed to work well with existing versions of YANG and allow for artifact authors to migrate to this scheme, it is not expected that all revisions of a given YANG artifact will have a semantic version label. For example, the first revision of a module or submodule may have been produced before this scheme was available.

YANG packages that make use of this semantic versioning scheme will have their semantic version as the value of the "revision_label" property.

As stated above, the YANG semver version number is expressed as a string of the form: 'X.Y.Z_COMPAT'; where X, Y, and Z each represent non-negative integers smaller than 2147483647 without leading zeroes, and _COMPAT represents an optional suffix of either "_compatible" or "_non_compatible".

- * 'X' is the MAJOR version. Changes in the MAJOR version number indicate changes that are non-backwards-compatible to versions with a lower MAJOR version number.
- * 'Y' is the MINOR version. Changes in the MINOR version number indicate changes that are backwards-compatible to versions with the same MAJOR version number, but a lower MINOR version number and no PATCH "_compatible" or "_non_compatible" modifier.

- * `'Z_COMPAT'` is the PATCH version and modifier. Changes in the PATCH version number can indicate editorial, backwards-compatible, or non-backwards-compatible changes relative to versions with the same MAJOR and MINOR version numbers, but lower PATCH version number, depending on what form modifier `"_COMPAT"` takes:
 - If the modifier string is absent, the change represents an editorial change. An editorial change is defined to be a change in the YANG artifact's content that does not affect the semantic meaning or functionality provided by the artifact in any way. Some examples include correcting a spelling mistake in the description of a leaf within a YANG module or submodule, non-significant whitespace changes (e.g. realigning description statements, or changing indentation), or changes to YANG comments. Note: restructuring how a module uses, or does not use, submodules is treated as an editorial level change on the condition that there is no change in the module's semantic behavior due to the restructuring.
 - If, however, the modifier string is present, the meaning is described below:
 - `"_compatible"` - the change represents a backwards-compatible change
 - `"_non_compatible"` - the change represents a non-backwards-compatible change

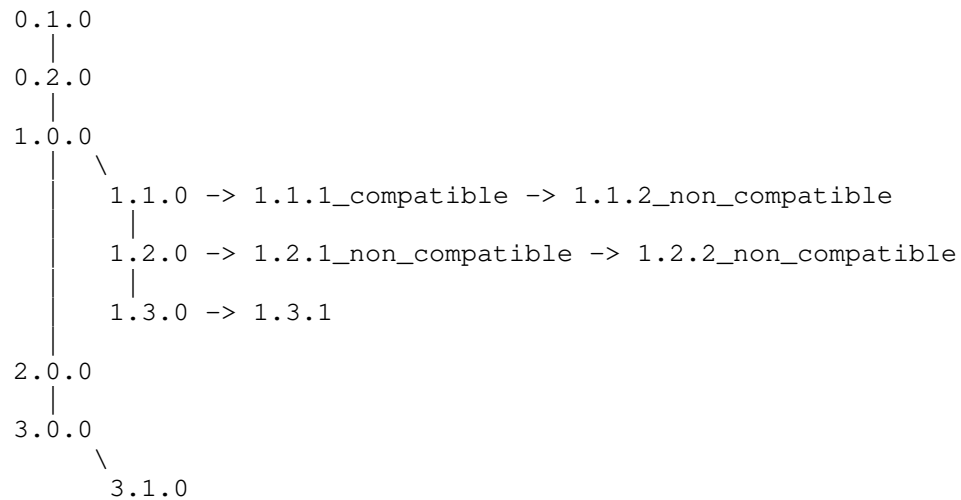
The YANG artifact name and YANG semantic version number uniquely identify a revision of said artifact. There MUST NOT be multiple instances of a YANG artifact definition with the same name and YANG semantic version number but different content (and in the case of modules and submodules, different revision dates).

There MUST NOT be multiple versions of a YANG artifact that have the same MAJOR, MINOR and PATCH version numbers, but different patch modifier strings. E.g., artifact version `"1.2.3_non_compatible"` MUST NOT be defined if artifact version `"1.2.3"` has already been defined.

3.2.1. Examples for YANG semantic version numbers

The following diagram and explanation illustrates how YANG semantic version numbers work.

Example YANG semantic version numbers for an example artifact:



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

0.1.0 - first beta module version

0.2.0 - second beta module version (with NBC changes)

1.0.0 - first release (may have NBC changes from 0.2.0)

1.1.0 - added new functionality, leaf "foo" (BC)

1.2.0 - added new functionality, leaf "baz" (BC)

1.3.0 - improve existing functionality, added leaf "foo-64" (BC)

1.3.1 - improve description wording for "foo-64" (Editorial)

1.1.1_compatible - backport "foo-64" leaf to 1.1.x to avoid implementing "baz" from 1.2.0 (BC)

2.0.0 - change existing model for performance reasons, e.g. re-key list (NBC)

1.1.2_non_compatible - NBC point bug fix, not required in 2.0.0 due to model changes (NBC)

3.0.0 - NBC bugfix, rename "baz" to "bar"; also add new BC leaf "wibble"; (NBC)

1.2.1_non_compatible - backport NBC fix, changing "baz" to "bar"

1.2.2_non_compatible - backport "wibble". This is a BC change but "non_compatible" modifier is sticky.

3.1.0 - introduce new leaf "wobble" (BC)

The partial ordering relationships based on the semantic versioning numbers can be defined as follows:

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 2.0.0 < 3.0.0 < 3.1.0

1.0.0 < 1.1.0 < 1.1.1_compatible < 1.1.2_non_compatible

1.0.0 < 1.1.0 < 1.2.0 < 1.2.1_non_compatible <
1.2.2_non_compatible

There is no ordering relationship between 1.1.1_non_compatible and either 1.2.0 or 1.2.1_non_compatible, except that they share the common ancestor of 1.1.0.

Looking at the version number alone, the module definition in 2.0.0 does not necessarily contain the contents of 1.3.0. However, the module revision history in 2.0.0 may well indicate that it was edited from module version 1.3.0.

3.3. YANG Semantic Version Update Rules

When a new revision of an artifact is produced, then the following rules define how the YANG semantic version number for the new artifact revision is calculated, based on the changes between the two artifact revisions, and the YANG semantic version number of the base artifact revision from which the changes are derived.

The following four rules specify the RECOMMENDED, and REQUIRED minimum, update to a YANG semantic version number:

1. If an artifact is being updated in a non-backwards-compatible way, then the artifact version "X.Y.Z[_compatible|_non_compatible]" SHOULD be updated to "X+1.0.0" unless that version has already been used for this artifact but with different content, in which case the artifact version "X.Y.Z+1_non_compatible" SHOULD be used instead.
2. If an artifact is being updated in a backwards-compatible way, then the next version number depends on the format of the current version number:

- i "X.Y.Z" - the artifact version SHOULD be updated to "X.Y+1.0", unless that version has already been used for this artifact but with different content, when the artifact version SHOULD be updated to "X.Y.Z+1_compatible" instead.
 - ii "X.Y.Z_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_compatible".
 - iii "X.Y.Z_non_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_non_compatible".
3. If an artifact is being updated in an editorial way, then the next version number depends on the format of the current version number:
 - i "X.Y.Z" - the artifact version SHOULD be updated to "X.Y.Z+1"
 - ii "X.Y.Z_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_compatible".
 - iii "X.Y.Z_non_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_non_compatible".
4. YANG artifact semantic version numbers beginning with 0, i.e., "0.X.Y", are regarded as beta definitions and need not follow the rules above. Either the MINOR or PATCH version numbers may be updated, regardless of whether the changes are non-backwards-compatible, backwards-compatible, or editorial. See Section 5 for more details on using this notation during module and submodule development.
5. XXX - Add some text about pre-release labels, or perhaps as a rule 5 above.

Although artifacts SHOULD be updated according to the rules above, which specify the recommended (and minimum required) update to the version number, the following rules MAY be applied when choosing a new version number:

1. An artifact author MAY update the version number with a more significant update than described by the rules above. For example, an artifact could be given a new MAJOR version number (i.e., X+1.0.0), even though no non-backwards-compatible changes have occurred, or an artifact could be given a new MINOR version number (i.e., X.Y+1.0) even if the changes were only editorial.

2. An artifact author MAY skip version numbers. That is, an artifact's revision history could be 1.0.0, 1.1.0, and 1.3.0 where 1.2.0 is skipped. Note that skipping versions has an impact when importing modules by revision-or-derived. See Section 4 for more details on importing modules with revision-label version gaps.

Although YANG Semver always indicates when a non-backwards-compatible, or backwards-compatible change may have occurred to a YANG artifact, it does not guarantee that such a change has occurred, or that consumers of that YANG artifact will be impacted by the change. Hence, tooling, e.g., [I-D.ietf-netmod-yang-schema-comparison] , also plays an important role for comparing YANG artifacts and calculating the likely impact from changes.

[I-D.ietf-netmod-yang-module-versioning] defines the "rev:nbc-changes" extension statement to indicate where non-backwards-compatible changes have occurred in the module revision history. If a revision entry in a module's revision history includes the "rev:nbc-changes" statement then that MUST be reflected in any YANG Semver version associated with that revision. However, the reverse does not necessarily hold, i.e., if the MAJOR version has been incremented it does not necessarily mean that a "rev:nbc-changes" statement would be present.

3.4. Examples of the YANG Semver Label

3.4.1. Example Module Using YANG Semver

Below is a sample YANG module that uses the YANG semver revision label based on the rules defined in this document.

```
module example-versioned-module {
  yang-version 1.1;
  namespace "urn:example:versioned:module";
  prefix "exvermod";
  rev:revision-label-scheme "yangver:yang-semver";

  import ietf-yang-revisions { prefix "rev"; }
  import ietf-yang-semver { prefix "yangver"; }

  description
    "to be completed";

  revision 2018-02-28 {
    description "Added leaf 'wobble'";
    rev:revision-label "3.1.0";
  }
}
```

```
}

revision 2017-12-31 {
  description "Rename 'baz' to 'bar', added leaf 'wibble'";
  rev:revision-label "3.0.0";
  rev:nbc-changes;
}

revision 2017-10-30 {
  description "Change the module structure";
  rev:revision-label "2.0.0";
  rev:nbc-changes;
}

revision 2017-08-30 {
  description "Clarified description of 'foo-64' leaf";
  rev:revision-label "1.3.1";
}

revision 2017-07-30 {
  description "Added leaf foo-64";
  rev:revision-label "1.3.0";
}

revision 2017-04-20 {
  description "Add new functionality, leaf 'baz'";
  rev:revision-label "1.2.0";
}

revision 2017-04-03 {
  description "Add new functionality, leaf 'foo'";
  rev:revision-label "1.1.0";
}

revision 2017-04-03 {
  description "First release version.";
  rev:revision-label "1.0.0";
}

// Note: semver rules do not apply to 0.X.Y labels.

revision 2017-01-30 {
  description "NBC changes to initial revision";
  semver:module-version "0.2.0";
}

revision 2017-01-26 {
  description "Initial module version";
```

```
    semver:module-version "0.1.0";  
  }  
  
  //YANG module definition starts here
```

3.4.2. Example of Package Using YANG Semver

Below is an example YANG package that uses the semver revision label based on the rules defined in this document.

```
{  
  "ietf-yang-instance-data:instance-data-set": {  
    "name": "example-yang-pkg",  
    "target-ptr": "TBD",  
    "timestamp": "2018-09-06T17:00:00Z",  
    "description": "Example IETF package definition",  
    "content-data": {  
      "ietf-yang-package:yang-package": {  
        "name": "example-yang-pkg",  
        "version": "1.3.1",  
        ...  
      }  
    }  
  }
```

4. Import Module by Semantic Version

[I-D.ietf-netmod-yang-module-versioning] allows for imports to be done based on a module or a derived revision of a module. The `rev:revision-or-derived` statement can specify either a revision date or a revision label. When importing by semver, the YANG semver revision label value MAY be used as an argument to `rev:revision-or-derived`. When used as such, any module which has that semver label as its latest revision label or has that label in its revision history can be used to satisfy the import requirement. For example:

```
import example-module {  
  rev:revision-or-derived "3.0.0";  
}
```

Note: the import lookup does not stop when a non-backward-compatible change is encountered. That is, if module B imports a module A at or derived from version 2.0.0, resolving that import will pass through a revision of module A with version 2.1.0_non_compatible in order to determine if the present instance of module A derives from 2.0.0.

If an import by revision-or-derived cannot locate the specified revision-label in a given module's revision history, that import will fail. This is noted in the case of version gaps. That is, if a module's history includes 1.0.0, 1.1.0, and 1.3.0, an import from revision-or-derived at 1.2.0 will be unable to locate the specified revision entry and thus the import cannot be satisfied.

5. Guidelines for Using Semver During Module Development

This section and the IETF-specific sub-section below provides YANG semver-specific guidelines to consider when developing new YANG modules. As such this section updates [RFC8407] .

Development of a brand new YANG module or submodule outside of the IETF that uses YANG semver as its revision-label scheme SHOULD begin with a 0 for the MAJOR version component. This allows the module or submodule to disregard strict semver rules with respect to non-backwards-compatible changes during its initial development. However, module or submodule developers MAY choose to use the semver pre-release syntax instead with a 1 for the MAJOR version component. For example, an initial module or submodule revision-label might be either 0.0.1 or 1.0.0-alpha.1. If the authors choose to use the 0 MAJOR version component scheme, they MAY switch to the pre-release scheme with a MAJOR version component of 1 when the module or submodule is nearing initial release (e.g., a module's or submodule's revision label may transition from 0.3.0 to 1.0.0-beta.1 to indicate it is more mature and ready for testing).

When using pre-release notation, the format MUST include at least one alphabetic component and MUST end with a '.' and then one or more digits. These alphanumeric components will be used when deciding pre-release precedence. The following are examples of valid pre-release versions

1.0.0-alpha.1

1.0.0-alpha.3

2.1.0-beta.42

3.0.0-202007.rc.1

When developing a new revision of an existing module or submodule using the YANG semver revision-label scheme, the intended target semver version MUST be used along with pre-release notation. For example, if a released module or submodule which has a current revision-label of 1.0.0 is being modified with the intent to make non-backwards-compatible changes, the first development MAJOR version

component must be 2 with some pre-release notation such as `-alpha.1`, making the version `2.0.0-alpha.1`. That said, every publicly available release of a module or submodule MUST have a unique YANG semver revision-label (where a publicly available release is one that could be implemented by a vendor or consumed by an end user). Therefore, it may be prudent to include the year or year and month development began (e.g., `2.0.0-201907-alpha.1`). As a module or submodule undergoes development, it is possible that the original intent changes. For example, a `1.0.0` version of a module or submodule that was destined to become `2.0.0` after a development cycle may have had a scope change such that the final version has no non-backwards-compatible changes and becomes `1.1.0` instead. This change is acceptable to make during the development phase so long as pre-release notation is present in both versions (e.g., `2.0.0-alpha.3` becomes `1.1.0-alpha.4`). However, on the next development cycle (after `1.1.0` is released), if again the new target release is `2.0.0`, new pre-release components must be used such that every revision-label for a given module or submodule MUST be unique throughout its entire lifecycle (e.g., the first pre-release version might be `2.0.0-202005-alpha.1` if keeping the same year and month notation mentioned above).

5.1. Pre-release Version Precedence

As a module or submodule is developed, the scope of the work may change. That is, while a ratified module or submodule with revision-label `1.0.0` is initially intended to become `2.0.0` in its next ratified version, the scope of work may change such that the final version is `1.1.0`. During the development cycle, the pre-release versions could move from `2.0.0-some-pre-release-tag` to `1.1.0-some-pre-release-tag`. This downwards changing of version numbers makes it difficult to evaluate semver rules between pre-release versions. However, taken independently, each pre-release version can be compared to the previously ratified version (e.g., `1.1.0-some-pre-release-tag` and `2.0.0-some-pre-release-tag` can each be compared to `1.0.0`). Module and submodule developers SHOULD maintain only one revision statement in a pre-released module or submodule that reflects the latest revision. IETF authors MAY choose to include an appendix in the associated draft to track overall changes to the module or submodule.

5.2. YANG Semver in IETF Modules

All published IETF modules and submodules MUST use YANG semantic versions for their revision-labels. For IETF YANG modules and submodules that have already been published, revision labels MUST be retrospectively applied to all existing revisions when the next new revision is created, starting at version "1.0.0" for the initial published revision, and then incrementing according to the YANG Semver version rules specified in Section 3.3 .

Net new module or submodule development within the IETF SHOULD begin with the 0 MAJOR number scheme as described above. When revising an existing IETF module or submodule, the revision-label MUST use the target (i.e., intended) MAJOR and MINOR version components with a 0 PATCH version component. If the intended ratified release will be non-backward-compatible with the current ratified release, the MINOR version component MUST be 0.

All IETF modules and submodules in development MUST use the whole document name as a pre-release version string, including the current document revision. For example, if a module or submodule which is currently released at version 1.0.0 is being revised to include non-backwards-compatible changes in draft-user-netmod-foo, its development revision-labels MUST include 2.0.0-draft-user-netmod-foo followed by the document's revision (e.g., 2.0.0-draft-user-netmod-foo-02). This will ensure each pre-release version is unique across the lifecycle of the module or submodule. Even when using the 0 MAJOR version for initial module or submodule development (where MINOR and PATCH can change), appending the draft name as a pre-release component helps to ensure uniqueness when there are perhaps multiple, parallel efforts creating the same module or submodule.

If a module or submodule is being revised and the original module or submodule never had a revision-label (i.e., you wish to start using YANG semver in future module or submodule revisions), choose a semver value that makes the most sense based on the module's or submodule's history. For example, if a module or submodule started out in the pre-NMDA ([RFC8342]) world, and then had NMDA support added without removing any legacy "state" branches -- and you are looking to add additional new features -- a sensible choice for the target YANG semver would be 1.2.0 (since 1.0.0 would have been the initial, pre-NMDA release, and 1.1.0 would have been the NMDA revision).

See Appendix A for a detailed example of IETF pre-release versions.

6. YANG Module

This YANG module contains the typedef for the YANG semantic version.

```
<CODE BEGINS> file "ietf-yang-semver@2020-06-30.yang"
module ietf-yang-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-semver";
  prefix yangver;
  rev:revision-label-scheme "yang-semver";

  import ietf-yang-revisions {
    prefix rev;
  }

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

    Author:   Joe Clarke
              <mailto:jclarke@cisco.com>";
  description
    "This module provides type and grouping definitions for YANG
    packages.

    Copyright (c) 2020 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 2020-06-30 {
    rev:revision-label "1.0.0-draft-ietf-netmod-yang-semver-01";
    description
```

```

        "Initial revision";
    reference
        "RFC XXXX: YANG Semantic Versioning.";
}

/*
 * Identities
 */

identity yang-semver {
    base rev:revision-label-scheme-base-identity;
    description
        "The revision-label scheme corresponds to the YANG semver scheme
        which is defined by the pattern in the 'version' typedef below.
        The rules governing this revision-label scheme are defined in the
        reference for this identity.";
    reference
        "RFC XXXX: YANG Semantic Versioning.";
}

/*
 * Typedefs
 */

typedef version {
    type string {
        pattern '\d+[.]\d+[.]\d+(_(non_)?compatible)?(-[\w\d.]+)?([+][\w\d\.]+'
)?';
    }
    description
        "Represents a YANG semantic version number. The rules governing the
        use of this revision label scheme are defined in the reference for
        this typedef.";
    reference
        "RFC XXXX: YANG Semantic Versioning.";
}
}
<CODE ENDS>

```

7. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The design team consists of the following members whom have worked on the YANG versioning project:

- * Balazs Lengyel
- * Benoit Claise

- * Ebben Aries
- * Jason Sterne
- * Joe Clarke
- * Juergen Schoenwaelder
- * Mahesh Jethanandani
- * Michael (Wangzitao)
- * Qin Wu
- * Reshad Rahman
- * 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 based on their own [openconfigsemver] . We would like thank both Anees Shaikh and Rob Shakir for their input into this problem space.

8. Security Considerations

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

9. IANA Considerations

9.1. YANG Module Registrations

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

Name: ietf-yang-semver

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

Prefix: yangver

Reference: [RFCXXXX]

9.2. Guidance for YANG Semver in IANA maintained YANG modules and submodules

Note for IANA (to be removed by the RFC editor): Please check that the registries and IANA YANG modules and submodules are referenced in the appropriate way.

IANA is responsible for maintaining and versioning some YANG modules and submodules, e.g., `iana-if-types.yang` [`IfTypeYang`] and `iana-routing-types.yang` [`RoutingTypesYang`] .

In addition to following the rules specified in the IANA Considerations section of [I-D.ietf-netmod-yang-module-versioning] , IANA maintained YANG modules and submodules MUST also include a YANG Semver revision label for all new revisions, as defined in Section 3 .

The YANG Semver version associated with the new revision MUST follow the rules defined in Section 3.3 .

Note: For IANA maintained YANG modules and submodules that have already been published, revision labels MUST be retrospectively applied to all existing revisions when the next new revision is created, starting at version "1.0.0" for the initial published revision, and then incrementing according to the YANG Semver version rules specified in Section 3.3 .

Most changes to IANA maintained YANG modules and submodules are expected to be backwards-compatible changes and classified as MINOR version changes. The PATCH version may be incremented instead when only editorial changes are made, and the MAJOR version would be incremented if non-backwards-compatible major changes are made.

Given that IANA maintained YANG modules and submodules are versioned with a linear history, it is anticipated that it should not be necessary to use the "_compatible" or "_non_compatible" modifiers to the "Z_COMPAT" version element.

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>>.

- [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>>.
- [I-D.ietf-netmod-yang-module-versioning]
Wilton, R., Rahman, R., Lengyel, B., Clarke, J., Sterne, J., Claise, B., and K. D'Souza, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-02, 22 February 2021, <<https://tools.ietf.org/html/draft-ietf-netmod-yang-module-versioning-02>>.

10.2. Informative References

- [I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <<https://tools.ietf.org/html/draft-clacla-netmod-yang-model-update-06>>.
- [I-D.ietf-netmod-yang-packages]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-01, 2 November 2020, <<https://tools.ietf.org/html/draft-ietf-netmod-yang-packages-01>>.
- [I-D.ietf-netmod-yang-schema-comparison]
Wilton, R., "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-01, 2 November 2020, <<https://tools.ietf.org/html/draft-ietf-netmod-yang-schema-comparison-01>>.
- [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>>.
- [openconfigsemver]
"Semantic Versioning for Openconfig Models", <<http://www.openconfig.net/docs/semver/>>.

```
[semver]    "Semantic Versioning 2.0.0 (text from June 19, 2020)",
            <https://github.com/semver/semver/
            blob/8b2e8eec394948632957639dfa99fc7ec6286911/semver.md>.

[IfTypeYang]
            "iana-if-type YANG Module",
            <https://www.iana.org/assignments/iana-if-type/iana-if-
            type.xhtml>.

[RoutingTypesYang]
            "iana-routing-types YANG Module",
            <https://www.iana.org/assignments/iana-routing-types/iana-
            routing-types.xhtml>.
```

Appendix A. Example IETF Module Development

Assume a new YANG module is being developed in the netmod working group in the IETF. Initially, this module is being developed in an individual internet draft, draft-jdoe-netmod-example-module. The following represents the initial version tree (i.e., value of revision-label) of the module as it's being initially developed.

Version lineage for initial module development:

```
0.0.1-draft-jdoe-netmod-example-module-00
|
0.1.0-draft-jdoe-netmod-example-module-01
|
0.2.0-draft-jdoe-netmod-example-module-02
|
0.2.1-draft-jdoe-netmod-example-module-03
```

At this point, development stabilizes, and the workgroup adopts the draft. Thus now the draft becomes draft-ietf-netmod-example-module. The initial pre-release lineage continues as follows.

Continued version lineage after adoption:

```
1.0.0-draft-ietf-netmod-example-module-00
|
1.0.0-draft-ietf-netmod-example-module-01
|
1.0.0-draft-ietf-netmod-example-module-02
```

At this point, the draft is ratified and becomes RFC12345 and the YANG module version number becomes 1.0.0.

A time later, the module needs to be revised to add additional capabilities. Development will be done in a backwards-compatible way. Two new individual drafts are proposed to go about adding the capabilities in different ways: draft-jdoe-netmod-exmod-enhancements and draft-jadoe-netmod-exmod-changes. These are initially developed in parallel with the following versions.

Parallel development for next module revision:

```
1.1.0-draft-jdoe-netmod-exmod-enhancements-00 || 1.1.0-draft-jadoe-netmod-e
xmod-changes-00
|
1.1.0-draft-jdoe-netmod-exmod-enhancements-01 || 1.1.0-draft-jadoe-netmod-e
xmod-changes-01
```

At this point, the WG decides to merge some aspects of both and adopt the work in jadoe's draft as draft-ietf-netmod-exmod-changes. A single version lineage continues.

```
1.1.0-draft-ietf-netmod-exmod-changes-00
|
1.1.0-draft-ietf-netmod-exmod-changes-01
|
1.1.0-draft-ietf-netmod-exmod-changes-02
|
1.1.0-draft-ietf-netmod-exmod-changes-03
```

The draft is ratified, and the new module version becomes 1.1.0.

Authors' Addresses

Benoit Claise
Huawei

Email: benoit.claise@huawei.com

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

Reshad Rahman
Cisco Systems, Inc.

Email: rrahman@cisco.com

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Balazs Lengyel
Ericsson
1117 Budapest
Magyar Tudosok Korutja
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

NETMOD
Internet-Draft
Updates: RFC6241, RFC8040, RFC8342 (if approved)
Intended status: Standards Track
Expires: 28 April 2022

Q. Ma, Ed.
C. Feng
Q. Wu
Huawei
25 October 2021

System-defined Configuration
draft-ma-netmod-with-system-00

Abstract

This document updates NMDA [RFC 8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To support non-NMDA servers, a "with-system" parameter has been defined to return <running> and system-defined configuration combined. The solution enables clients to reference nodes defined in <system>, overwrite values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 28 April 2022.

Copyright Notice

Copyright (c) 2021 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. Requirements Language	4
1.3. Updates to RFC 6241	4
1.4. Updates to RFC 8040	5
2. Kinds of System Configuration	5
2.1. Immediately-Active	5
2.2. Conditionally-Active	6
2.3. Inactive-Until-Referenced	6
3. Static Characteristics	6
3.1. Read-only to Clients	6
3.2. May Change via Software Upgrades	6
3.3. No Impact to <operational>	7
4. Dynamic Behavior	7
4.1. Conceptual Model	7
4.2. Modifying (overriding) system configuration	8
4.3. Explicit declaration of system configuration	8
4.4. Examples	9
4.4.1. Modifying A System-instantiated Leaf's Value	9
4.4.2. Configuring Descendant Nodes of A System-defined Node	11
4.4.3. Declaring A System-defined Node in <running> Explicitly	13
5. Discovering System Configuration	13
5.1. The "with-system" Query Parameter	14
5.2. The <system> Configuration Datastore	14
6. The "ietf-netconf-with-system" Module	15
6.1. Data Model Overview	15
6.2. Example Usage	16
6.3. YANG Module	17
7. The "ietf-system-datastore" Module	19
7.1. Data Model Overview	19
7.2. Example Usage	20
7.3. YANG Module	20
8. IANA Considerations	21
8.1. The "IETF XML" Registry	21
8.2. The "YANG Module Names" Registry	22

9. Security Considerations	22
9.1. Regarding the "ietf-netconf-with-system" YANG Module . . .	22
9.2. Regarding the "ietf-system-datastore" YANG Module	23
10. Contributors	23
Acknowledgements	23
References	23
Normative References	23
Informative References	24
Appendix A. Key Use Cases	24
A.1. Device Powers On	24
A.2. Client Commits Configuration	25
A.3. Operator Installs Card into a Chassis	26
Appendix B. Changes between Revisions	27
Appendix C. Open Issues tracking	28
Authors' Addresses	28

1. Introduction

NMDA Architecture [RFC8342] defines system configuration as the configuration that is supplied by the device itself and should be present in <operational> when it is in use.

However, there is a desire to enable a server to better document the system configuration. Clients can benefit from a standard mechanism to see what system configuration is available in a server.

In some cases, a client or offline tool may consider the configuration in <running> or <intended> invalid due to references (e.g. leafref) to system configuration data that isn't returned when the datastore is read. The server may accept a configuration (i.e. by internally merging the client specified contents of <running> with the server-provided system configuration and validating the result), but the client or offline tool would consider the datastore contents as invalid.

Having to copy the entire contents of the system configuration into <running> should be avoided or reduced when possible.

In some other cases, configuration of descendant nodes of system defined configuration needs to be supported. For example, the system configuration may contain an almost empty physical interface, while the client needs to be able to add, modify, remove a number of descendant nodes. Some descendant nodes may not be modifiable (e.g. "name" and "type" set by the system).

In all cases, the clients should have control over the configurations ,i.e., read-back of <running> should contain only what was explicitly set by clients.

This document updates NMDA [RFC 8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To support non-NMDA servers, a "with-system" parameter has been defined to return <running> and system-defined configuration combined. The solution enables clients to reference nodes defined in <system>, overwrite values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

1.1. Terminology

This document assumes that the reader is familiar with the contents of [RFC6241], [RFC7950], [RFC8342], [RFC8407], and [RFC8525] and uses terminologies from those documents.

The following terms are defined in this document as follows:

System configuration: Configuration that is provided by the system itself [RFC8342].

Conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, <system>, and <intended>. These datastores share a common datastore schema, and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.

System configuration datastore: A configuration datastore holding the complete configuration provided by the system itself. This datastore is referred to as "<system>".

1.2. Requirements Language

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

1.3. Updates to RFC 6241

The <get> and <get-config> RPC operations defined in [RFC6241] are augmented to accept additional new input parameter "with-system" which carries no value. The retrieval of implicit hidden system configuration in <running> can be used through <get> or <get-config> operation with the presence of "with-system" parameter.

The implicit hidden system configuration will contain all three types of system configurations defined in Section 2.

Note that the <get-data> RPC operation defined in [RFC8526] can also be augmented to retrieve the system configuration from <running>. But not sure whether the new client only supports <get-data> operation or supports both <get-config> operation and <get-data> operation.

1.4. Updates to RFC 8040

This document extends Section 4.8 of [RFC8040] to add a new query parameter "with-system".

The "with-system" parameter controls whether implicitly hidden system configuration will be returned in the reply. This parameter is only allowed with no values carried. If this parameter has any unexpected value, then a "400 Bad Request" status-line is returned.

Name	Methods	Description
with-system	GET, HEAD	indicates that the implicitly hidden system configuration should be returned. If not specified, then no implicitly hidden system configuration should be returned. This parameter can be given in any order.

2. Kinds of System Configuration

There are three types of system configurations: immediately-activated system configuration, conditionally-activated system configuration and inactivated-until-referenced system configuration.

2.1. Immediately-Active

Immediately-active system configurations are those applied and active immediately (e.g., a loop-back interface) , irrespective of physical resource present or not, a special functionality enabled or not.

2.2. Conditionally-Active

System configurations which are provided and activated based on specific conditions being met in a system, e.g., if a physical resource is present (e.g., insert interface card), the system will automatically detect it and load pre-provisioned configuration; when the physical resource is not present (remove interface card), the system configuration will be automatically cleared. Another example is when a special functionality is enabled, e.g., when QoS function is enabled, QoS policies are automatically created by the system.

2.3. Inactive-Until-Referenced

There are some predefined objects (e.g., application ids, anti-x signatures, trust anchor certs, etc) as a convenience for the clients. The clients can also define their own data objects for their unique requirements. Inactive-until-referenced system configurations are not applied and active immediately but only after they are referenced by client defined configuration.

3. Static Characteristics

3.1. Read-only to Clients

From the clients' perspective, the contents of the <system> datastore are read-only. There is no way to delete system configuration from a server. Any deletable system-provided configuration must be defined in <factory-default> [RFC 8808], which is used to initialize <running> when the device is first-time powered on or reset to its factory default condition.

3.2. May Change via Software Upgrades

System configuration MAY change dynamically, e.g., depending on factors like during device upgrade or system-controlled resources (e.g., HW available). In some implementations, when QoS function is enabled, QoS-related predefined policies are created by system. If the system configuration gets changed, YANG notification (e.g., "push-change-update" notification) [RFC8641] [RFC8639] [RFC6470] can be used to notify the client.

3.3. No Impact to <operational>

This work intends to have no impact to <operational>. As always, system configuration will appear in <operational> with "origin=system". This work enables a subset of those system generated nodes to be defined like configuration, i.e., made visible to clients in order for being referenced or configurable prior to present in <operational>. "Config false" nodes are completely out of scope, hence existing "config false" nodes are not impacted by this work.

4. Dynamic Behavior

4.1. Conceptual Model

This document introduces an optional datastore named "system" which is used to hold all three types of system configurations defined in Section 2.

When the device is powered on, immediately-activated system configuration will be provided and activated immediately but inactivated-until-referenced system configuration only becomes active if it is referenced by client defined configuration. While conditionally-activated system configuration will be created and immediately activated if the condition on system resources is met when the device is powered on or running.

All these system configuration will be implicitly hidden in the <running>, hence the client can retrieve them through standard operations defined in YANG-driven management protocols such as NETCONF and RESTCONF with a "with-system" query parameter. So that the client can get a merged view from the server.

If the <system> datastore exists, all above three types of system configurations will also go into <system>. Then the server will merge <running> and <system> to create <intended>, in which process, <running> MAY overwrite and/or extend <system>. If a server implements <intended>, <system> MUST be merged into <intended>.

When the client needs to configure the descendant nodes of system configuration(e.g., a physical interfaces), the ancestor system configuration needs to be configured in <running> explicitly.

4.2. Modifying (overriding) system configuration

In some cases, a server may allow some parts of system configuration to be modified. List keys in system configuration can't be changed by a client, but other descendant nodes in a list entry may be modifiable or non-modifiable. Leafs and leaf-lists outside of lists may also be modifiable or non-modifiable. Modification of system configuration is achieved by the client writing configuration to `<running>` that overrides the system configuration. Client configuration statements in `<running>` take precedence over system configuration nodes in `<system>` if the server allows the nodes to be modified. If a system configuration node is non-modifiable, then writing a value for that node in `<running>` returns an error.

A server may also allow a client to add data nodes to a list entry in `<system>` by writing those additional nodes in `<running>`. Those additional data nodes may not exist in `<system>` (i.e. an *addition* rather than an override).

While modifying (overriding) system configuration nodes may be supported by a server, there is no mechanism for deleting a system configuration node. A "mandatory true" leaf, for example, may have a value in `<system>` which can be modified (overridden) by a client setting that leaf to a value in `<running>`. But the leaf could not be deleted.

Comment 1: What if `<System>` contains a set of values for a leaf-list, and a client configures another set of values for that leaf-list in `<running>`, will the set of values in `<running>` completely replace the set of values in `<system>`? Or the two sets of values are merged together?

Comment 2: how "ordered-by user" lists and leaf-lists are merged? Do the `<running>` values go before or after, or is this a case where a full-replace is needed.

4.3. Explicit declaration of system configuration

In addition to modifying system configuration, and adding nodes to lists in system configuration as described above, a client can also explicitly declare system configuration nodes in `<running>` with the same values as in `<system>`. When a client configures a node (list entry, leaf, etc) in `<running>` that matches the same node & value in `<system>`, then that node becomes part of `<running>`. A read of `<running>` returns those explicitly configured nodes.

This explicit configuration of system configuration in <running> can be useful, for example, when an operator's workflow requires a client or offline tool to see the <running> configuration as valid. The client can explicitly declare (i.e. configure in <running>) the list entries (with at least the keys) for any system configuration list entries that are referenced elsewhere in <running>. The client does not necessarily need to declare all the contents of the list entry (i.e. the descendant nodes) - only the parts that are required to make the <running> appear valid offline.

4.4. Examples

The examples within this document use the fictional interface YANG module defined in Appendix C.3 of [RFC8342]. In addition, a fictional QoS data model example is provided.

4.4.1. Modifying A System-instantiated Leaf's Value

In this subsection, we will use this fictional QoS data model:

```
container qos-policies {
  list policy {
    key "name";
    leaf name {
      type string;
    }
  }
  list queue {
    key "queue-id";
    description "Enter the queue list instance";
    leaf queue-id {
      type int32 {
        range "1..32";
      }
    }
    leaf maximum-burst-size {
      type int32 {
        range "0..100";
      }
    }
  }
}
```

Suppose a client creates a qos policy "my-policy" with 4 system instantiated queues(1~4). The Configuration of qos-policies is present in <system> as follows:

```
<qos-policies>
  <name>my-policy</name>
  <queue>
    <queue-id>1</queue-id>
    <maximum-burst-size>50</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>2</queue-id>
    <maximum-burst-size>60</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>3</queue-id>
    <maximum-burst-size>70</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>4</queue-id>
    <maximum-burst-size>80</maximum-burst-size>
  </queue>
</qos-policies>
```

A client modifies the value of maximum-burst-size to 55 in queue-id 1:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <qos-policies>
        <name>my-policy</name>
        <queue>
          <queue-id>1</queue-id>
          <maximum-burst-size>55</maximum-burst-size>
        </queue>
      </qos-policies>
    </config>
  </edit-config>
</rpc>
```

Then the configuration of qos-policies is present in <operational> as follows:

```
<qos-policies xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
              or:origin="or:intended">
  <name>my-policy</name>
  <queue>
    <queue-id>1</queue-id>
    <maximum-burst-size>55</maximum-burst-size>
  </queue>
  <queue or:origin="or:system">
    <queue-id>2</queue-id>
    <maximum-burst-size>60</maximum-burst-size>
  </queue>
  <queue or:origin="or:system">
    <queue-id>3</queue-id>
    <maximum-burst-size>70</maximum-burst-size>
  </queue>
  <queue or:origin="or:system">
    <queue-id>4</queue-id>
    <maximum-burst-size>80</maximum-burst-size>
  </queue>
</qos-policies>
```

4.4.2. Configuring Descendant Nodes of A System-defined Node

Suppose the system provides a loopback interface (named "lo0") with a default IPv4 address of "127.0.0.1" and a default IPv6 address of "::1".

The configuration of "lo0" interface is present in <system> as follows:

```
<interfaces>
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

The configuration of "lo0" interface is present in <operational> as follows:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
             or:origin="or:system">
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Later on, the client further configures the description node of a "lo0" interface as follows:

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces>
        <interface>
          <name>lo0</name>
          <description>loopback</description>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

Then the configuration of interface "lo0" is present in <operational> as follows:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
             or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address or:origin="or:system">::1</ip-address>
  </interface>
</interfaces>
```

4.4.3. Declaring A System-defined Node in <running> Explicitly

In the environment which offline validation of <running> is required, a client need to declare the system-defined configurations that are actually referenced. Here is an example of a client explicitly declaring "lo0" in <running>. The client configures a "lo0" interface only with the list key "name" as follows:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces>
        <interface>
          <name>lo0</name>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

A read-back of <running> should looks like:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces>
        <interface>
          <name>lo0</name>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

5. Discovering System Configuration

There are two ways to discover system configuration: a "with-system" query parameter and a <system> configuration datastore.

5.1. The "with-system" Query Parameter

As defined in Section 1.3 and Section 1.4, All the system configuration will be implicitly hidden in <running>, hence the client can retrieve them through standard operations defined in YANG-driven management protocols such as NETCONF and RESTCONF with a "with-system" parameter to get a merged view.

All servers MUST implement a "with-system" parameter.

5.2. The <system> Configuration Datastore

This section is not applicable to non-NMDA servers. NMDA servers SHOULD implement a <system> configuration datastore, and they SHOULD also implement the <intended> datastore, which can be used as an alternative to "with-system" parameter.

Following guidelines for defining datastores in the appendix A of [RFC8342], this document introduces a new optional datastore resource named 'system' that represents the system configuration. A device MAY implement the mechanism defined in this document without implementing the "system" datastore, which would only eliminate the ability to programmatically determine the system configuration.

- * Name: "system"
- * YANG modules: all
- * YANG nodes: all "config true" data nodes
- * 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, but may change itself by upgrades and/or when resource-conditions are met. The datastore can be read using the standard NETCONF/RESTCONF protocol operations.
- * Origin: This document does not define any new origin identity when it interacts with <intended> datastore and finally flows into <operational>. The "system" origin Metadata Annotation [RFC7952] is used to indicate the origin of a data item.

Comment: Should we define any new origin identity to indicate new source of system configuration datastore?

- * Protocols: YANG-driven management protocols, such as NETCONF and RESTCONF.

* Defining YANG module: "ietf-system".

The datastore's content is populated by the server and read-only to clients. Upon the content is created or changed, it will be merged into <intended> datastore. Unlike <factory-default>[RFC8808], it MAY change dynamically, e.g., depending on factors like during device upgrade or system-controlled resources(e.g., HW available) and the <system> datastore does not have to persist across reboots. <factory-reset> RPC operation defined in [RFC8088] can reset it to its factory default configuration without including configuration generated due to the system update or client-enabled functionality.

6. The "ietf-netconf-with-system" Module

6.1. Data Model Overview

This YANG module augments NETCONF <get> and <get-config> operation, which is designed to make implicitly hidden system configuration visible via a "with-system" parameter.

The following tree diagram [RFC8340] illustrates the "ietf-netconf-with-system" module:

```
module: ietf-netconf-with-system
  augment /nc:get-config/nc:input:
    +---w with-system?   empty
  augment /nc:get/nc:input:
    +---w with-system?   empty
```

The following tree diagram [RFC8340] illustrates "get" and "get-config" rpcs defined in "ietf-netconf" augmented by "ietf-netconf-with-system" module :

```

rpcs:
  +---x get-config
  |   +---w input
  |   |   +---w source
  |   |   |   +---w (config-source)
  |   |   |   |   +---:(candidate)
  |   |   |   |   |   +---w candidate?    empty {candidate}?
  |   |   |   |   |   +---:(running)
  |   |   |   |   |   |   +---w running?    empty
  |   |   |   |   |   +---:(startup)
  |   |   |   |   |   |   +---w startup?    empty {startup}?
  |   |   |   +---w filter?    <anyxml>
  |   |   +---w with-system?    empty
  |   +---ro output
  |   |   +---ro data?    <anyxml>
  +---x get
  |   +---w input
  |   |   +---w filter?    <anyxml>
  |   |   +---w with-system?    empty
  |   +---ro output
  |   |   +---ro data?    <anyxml>

```

6.2. Example Usage

This section gives an example of request/response pairs with and without the "with-system" query parameter. The YANG module used are shown in Appendix C.2 of [RFC8342].

Suppose the following data is added to <running>:

```

{
  "bgp": {
    "local-as": "64501",
    "peer-as": "64502",
    "peer": {
      "name": "2001:db8::2:3"
    }
  }
}

```

All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

REQUEST(without a "with-system" query parameter):

```

Target:/bgp
Query Parameter:
with-defaults: report-all

```

RESPONSE(both bgp/peer/local-as and bgp/peer/peer-as have default values for a peer. "local-port" leaf is not present in <running>):

```
{
  "bgp": {
    "local-as": "64501",
    "peer-as": "64502",
    "peer": {
      "name": "2001:db8::2:3",
      "local-as": "64501",
      "peer-as": "64502",
      "remote-port": "179",
      "state": "established"
    }
  }
}
```

REQUEST(with a "with-system" query parameter):

```
Target:/bgp
Query Parameter:
with-system
with-defaults: report-all
```

RESPONSE(local-port leaf value is supplied by the system):

```
{
  "bgp": {
    "local-as": "64501",
    "peer-as": "64502",
    "peer": {
      "name": "2001:db8::2:3",
      "local-as": "64501",
      "peer-as": "64502",
      "local-port": "60794",
      "remote-port": "179",
      "state": "established"
    }
  }
}
```

6.3. YANG Module

```
<CODE BEGINS>
file="ietf-netconf-with-system@2021-05-14.yang"
module ietf-netconf-with-system {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-with-system";
  prefix ncws;

  import ietf-netconf {
    prefix nc;
    reference
      "RFC 6241: Network Configuration Protocol (NETCONF)";
  }

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

  contact
    "WG Web:  <http://tools.ietf.org/wg/netmod/>
    WG List:  <mailto:netmod@ietf.org>";
  description
    "This module defines an extension to the NETCONF protocol
    that allows the NETCONF client to control how system configuration
    data are handled by the server in particular NETCONF operations.

    Copyright (c) 2010 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.";
  // RFC Ed.: replace XXXX with actual RFC number and remove this note

  revision 2021-05-14 {
    description
      "Initial version.";
    reference
      "RFC XXXX: System configuration Data handling Behavior";
  }

  augment /nc:get-config/nc:input {
    description " Allows the get-config operation to use
    with-system to retrieve the complete system configuration.";
  }
}
```

```

    leaf with-system {
        type empty ;
        description
            "Support system configuration retrieval on
            conventional configuration datastore. ";
    }
}

augment /nc:get/nc:input {
    description " Allows the get operation to use
    with-system to retrieve the complete system configuration.";
    leaf with-system {
        type empty ;
        description
            "Support system configuration retrieval on
            running datastore.";
    }
}
}
}
<CODE ENDS>

```

7. The "ietf-system-datastore" Module

7.1. Data Model Overview

This YANG module defines a new YANG identity named "system" that uses the "ds:datastore" identity defined in [RFC8342]. Note that no new origin identity is defined in this document, the "or:system" origin Metadata Annotation [RFC7952] is used to indicate the origin of a data item.

The following diagram illustrates the relationship amongst the "identity" statements defined in the "ietf-system-datastore" and "ietf-datastores" YANG modules

Identities:

```

+--- datastore
|
| +--- conventional
| |
| | +--- running
| | +--- candidate
| | +--- startup
| | +--- system
| | +--- intended
| +--- dynamic
| +--- operational

```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

7.2. Example Usage

This section gives an example of data retrieval from <system>.

Suppose the following data is added to <running>:

```
{
  "bgp": {
    "local-as": "64501",
    "peer-as": "64502",
    "peer": {
      "name": "2001:db8::2:3"
    }
  }
}
```

All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

REQUEST:

Datastore: <system>
Target:/bgp

RESPONSE("local-port" leaf value is supplied by the system):

```
{
  "bgp": {
    "peer": {
      "local-port": "60794"
    }
  }
}
```

7.3. YANG Module

```
<CODE BEGINS>
file="ietf-system-datastore@2021-05-14.yang"
module ietf-system-datastore {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-system-datastore";
  prefix sysds;

  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }
}
```

```
organization
  "IETF NETMDOD (Network Modeling) Working Group";

contact
  "WG Web:  <http://tools.ietf.org/wg/netmod/>
  WG List:  <mailto:netmod@ietf.org>";
description
  "This module defines a new YANG identity that uses the
  ds:datastore identity defined in [RFC8342].

  Copyright (c) 2010 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.";
// RFC Ed.: replace XXXX with actual RFC number and remove this note

revision 2021-05-14 {
  description

    "Initial version.";
  reference
    "RFC XXXX: System configuration Data handling Behavior";
}

identity system {
  base ds:conventional;
  description
    "This read-only datastore contains the complete configuration
    provided by the system itself.";
}
}
<CODE ENDS>
```

8. IANA Considerations

8.1. The "IETF XML" Registry

This document registers two XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-with-system

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-system-datastore

Registrant Contact: The IESG.

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

8.2. The "YANG Module Names" Registry

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020] .

```
name: ietf-netconf-with-system
prefix: ncws
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-with-system
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment
```

```
name: ietf-system-datastore
prefix: sys
namespace: urn:ietf:params:xml:ns:yang:ietf-system-datatstore
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment
```

9. Security Considerations

9.1. Regarding the "ietf-netconf-with-system" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and 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 Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

The security considerations for the base NETCONF protocol operations (see Section 9 of [RFC6241]) apply to the new extended RPC operations defined in this document.

9.2. Regarding the "ietf-system-datastore" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and 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 Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

10. Contributors

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf@chinatelecom.cn

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Acknowledgements

Thanks to Robert Wilton, Balazs Lengyel, Andy Bierman, Jan Lindbland, Juergen Schoenwaelder, Alex Clemm, Timothy Carey for reviewing, and providing important input to, this document.

References

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>>.

Informative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [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>>.
- [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>>.
- [RFC8808] Wu, Q., Lengyel, B., and Y. Niu, "A YANG Data Model for Factory Default Settings", RFC 8808, DOI 10.17487/RFC8808, August 2020, <<https://www.rfc-editor.org/info/rfc8808>>.

Appendix A. Key Use Cases

Following provides three use cases related to system-defined configuration lifecycle management. The simple interface data model defined in Appendix C.3 of [RFC8342] is used. For each use case, snippets of <running>, <system>, <intended> and <operational> are shown.

A.1. Device Powers On

<running>:

No configuration for lo0 appears in <running>;

<system>:

```
<interfaces>
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

<intended>:

```
<interfaces>
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

<operational>:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
  or:origin="or:system">
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

A.2. Client Commits Configuration

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point:

<running>:

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

<system>:

```
<interfaces>
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

<intended>:

```
<interfaces>
  <name>lo0</name>
  <ip-address>127.0.0.1</ip-address>
  <ip-address>::1</ip-address>
</interface>
<interface>
  <name>et-0/0/0</name>
  <description>Test interface</description>
</interface>
<interface>
</interface>
</interfaces>
```

<operational>:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
             or:origin="or:intended">
  <interface or:origin="or:system">
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

A.3. Operator Installs Card into a Chassis

<running>:

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

<system>:

```

<interfaces>
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <mtu>1500</mtu>
  </interface>
</interfaces>

```

<intended>:

```

<interfaces>
  <name>lo0</name>
  <ip-address>127.0.0.1</ip-address>
  <ip-address>::1</ip-address>
</interface>
<interface>
  <name>et-0/0/0</name>
  <description>Test interface</description>
  <mtu>1500</mtu>
</interface>
<interface>
</interface>
</interfaces>

```

<operational>:

```

<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
             or:origin="or:intended">
  <interface or:origin="or:system">
    <name or:origin>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
    <mtu or:origin="or:system">1500</mtu>
  </interface>
  <interface>
</interface>
</interfaces>

```

Appendix B. Changes between Revisions

v02 - v00

- * Restructure the document content based on input in the system defined configuration interim meeting.
- * Updates NMDA to define a read-only conventional configuration datastore called "system".
- * Retrieval of implicit hidden system configuration via <get><get-config> with "with-system" parameter to support non-NMDA servers.
- * Provide system defined configuration classification.
- * Define Static Characteristics and dynamic behavior for system defined configuration.
- * Separate "ietf-system-datastore" Module from "ietf-netconf-with-system" Module.
- * Provide usage examples for dynamic behaviors.
- * Provide usage examples for two YANG modules.
- * Provide three use cases related to system-defined configuration lifecycle management.
- * Classify the relation with <factory-default>.

Appendix C. Open Issues tracking

- * Backward compatibility: consider the communication between the server and the new client or the old client simultaneously.
- * Running always be valid? The client might need to understand how to merge if offline validation on running is used.
- * Immutable flag

Authors' Addresses

Qiufang Ma (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: maqiufang1@huawei.com

Feng Chong
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: frank.fengchong@huawei.com

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

Email: bill.wu@huawei.com

CCAMP Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 April 2022

C. Yu
I. Busi
Huawei Technologies
A. Guo
Futurewei Technologies
S. Belotti
Nokia
J-F. Bouquier
Vodafone
F. Peruzzini
TIM
O.G.d. Dios
Telefonica
V. Lopez
Nokia
25 October 2021

A YANG Data Model for Optical Network Inventory
draft-yg3bp-ccamp-optical-inventory-yang-00

Abstract

This document defines a YANG data model for optical network inventory data information.

The YANG data model presented in this document is intended to be used as the basis toward a generic YANG data model for network inventory data information which can be augmented, when required, with technology-specific (e.g., optical) inventory data, to be defined either in a future version of this document or in another document.

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

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 28 April 2022.

Copyright Notice

Copyright (c) 2021 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 and Notations	4
1.2. Tree Diagram	5
1.3. Prefix in Data Node Names	5
2. YANG Data Model for Optical Network Inventory	6
2.1. YANG Model Overview	6
3. Optical Network Inventory Tree Diagram	9
4. YANG Model for Optical Network Inventory	10
5. Manageability Considerations	16
6. Security Considerations	17
7. IANA Considerations	17
8. References	17
8.1. Normative References	17
8.2. Informative References	18
Acknowledgments	18
Authors' Addresses	18

1. Introduction

Network inventory management is a key component in operators' OSS architectures.

Network inventory is a fundamental functionality in network management and was specified many years ago. Given the emerging of data models and their deployment in operator's management and control systems, the traditional function of inventory management is also requested to be defined as a data model.

Network inventory management and monitoring is a critical part of ensuring the network stays healthy, well-planned, and functioning in the operator's network. Network inventory management allows the operator to keep track of what physical network devices are staying in the network including relevant software and hardware.

The network inventory management also helps the operator to know when to acquire new assets and what is needed, or to decommission old or faulty ones, which can help to improve network performance and capacity planning.

In [I-D.ietf-teas-actn-poi-applicability] a gap was identified regarding the lack of a YANG data model that could be used at ACTN MPI interface level to report whole/partial hardware inventory information available at PNC level towards north-bound systems (e.g., MDSC or OSS layer).

[RFC8345] initial goal was to make possible the augmentation of the YANG data model with network inventory data model but this was never developed and the scope was kept limited to network topology data only.

It is key for operators to drive the industry towards the use of a standard YANG data model for network inventory data instead of using vendors proprietary APIs (e.g., REST API).

In the ACTN architecture, this would bring also clear benefits at MDSC level for packet over optical integration scenarios since this would enable the correlation of the inventory information with the links information reported in the network topology model.

The intention is to define a generic YANG data model that would be as much as possible technology agnostic (valid for IP, optical and microwave networks) and that could be augmented, when required, to include some technology-specific inventory details.

[RFC8348] defines a YANG data model for the management of the hardware on a single server and therefore it is more applicable to the PNC South Bound Interface (SBI) towards the network elements rather than at the PNC MPI. However, the YANG data model defined in [RFC8348] has been used as a reference for defining the YANG network inventory data model.

For optical network inventory, the network inventory YANG data model should support the use cases (4a and 4b) and requirements defined in [ONF_TR-547], in order to guarantee a seamless integration at MDSC/OSS/orchestration layers.

The proposed YANG data model has been analysed to cover the requirements and use cases for Optical Network Inventory.

Being based on [RFC8348], this data model should be a good starting point toward a generic data model and applicable to any technology. However, further analysis of requirements and use cases is needed to extend the applicability of this YANG data model to other types of networks (IP and microwave) and to identify which aspects are generic and which aspects are technology-specific for optical networks.

This document defines one YANG module: `ietf-network-inventory.yang` (Section 4).

Note: review in future versions of this document the related modules, depending on the augmentation relationship.

The YANG data model defined in this document conforms to the Network Management Datastore Architecture [RFC8342].

1.1. Terminology and Notations

Refer to [RFC7446] and [RFC7581] for the key terms used in this document. The following terms are defined in [RFC7950] and are not redefined here:

- * client
- * server
- * augment
- * data model
- * data node

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

- * configuration data
- * state data

The terminology for describing YANG data models is found in [RFC7950].

TBD: Recap the concept of chassis/slot/component/board/... in [TMF-MTOSI].

Following terms are used for the representation of the hierarchies in the optical network inventory.

Network Element: a device installed on one or several shelves and can afford some specific transmission function independently.

Cabinet: a holder of the device and provides power supply for the device in it.

Chassis: a holder of the device installation.

Slot: a holder of the board.

Component: holders and equipments of the network element, including rack, shelf, slot, sub-slot, board and port.

Board/Card: a pluggable equipment on the network element and can afford a specific transmission function independently.

Port: an interface on board

1.2. Tree Diagram

A simplified graphical representation of the data model is used in Section 3 of this document. The meaning of the symbols in these diagrams is defined in [RFC8340].

1.3. Prefix in Data Node Names

In this document, names of data nodes and other data model objects are prefixed using the standard prefix associated with the corresponding YANG imported modules, as shown in the following table.

Prefix	Yang Module	Reference
ianahw	iana-hardware	[RFC8348]
ni	ietf-network-inventory	RFCXXX
yang	ietf-yang-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

RFC Editor Note: Please replace XXXX with the RFC number assigned to this document. Please remove this note.

2. YANG Data Model for Optical Network Inventory

2.1. YANG Model Overview

Based on TMF classification in [TMF-MTOSI], inventory objects can be divided into two groups, holder group and equipment group. The holder group contains rack, shelf, slot, sub-slot while the equipment group contains network-element, board and port. With the requirement of GIS and on-demand domain controller selection raised, the equipment room becomes a new inventory object to be managed besides TMF classification.

Logically, the relationship between these inventory objects can be described by Figure 1 below:

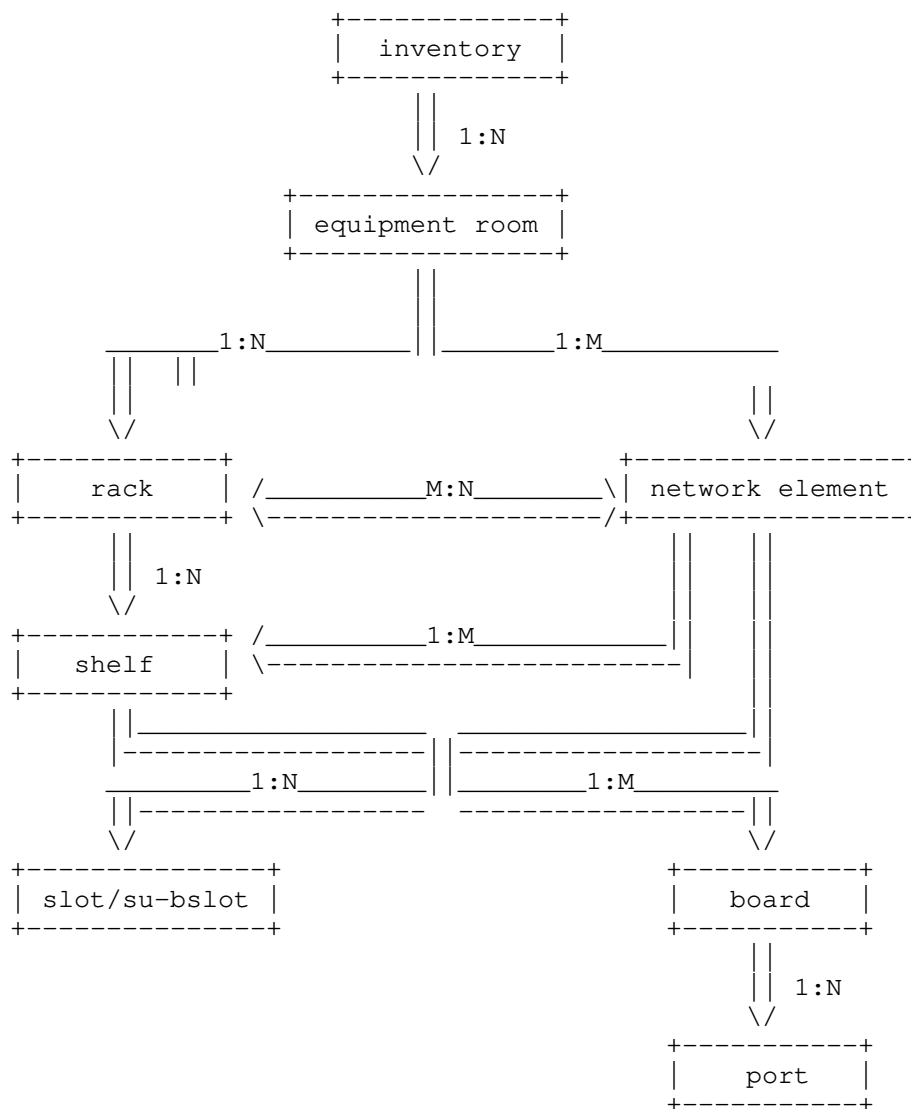


Figure 1: Relationship between inventory objects

In [RFC8348], rack, shelf, slot, sub-slot, board and port are defined as components of network elements with generic attributes.

While [RFC8348] is used to manage the hardware of a single server (e.g., a Network Element), the Network Inventory YANG data model is used to retrieve the network inventory information that a controller discovers from multiple Network Elements under its control.

However, the YANG data model defined in [RFC8348] has been used as a reference for defining the YANG network inventory data model. This approach can simplify the implementation of this network inventory model when the controller uses the YANG data model defined in [RFC8348] to retrieve the hardware configuration from the network elements under its control.

Note: review in future versions of this document which attributes from [RFC8348] are required also for network inventory and whether there are attributes not defined in [RFC8348] which are required for network inventory

Note: review in future versions of this document whether to re-use definitions from [RFC8348] or use schema-mount.

```

+--ro network-inventory
  +--ro equipment-rooms
    +--ro equipment-room* [uuid]
      +--ro uuid          yang:uuid
      .....
    +--ro rack* [uuid]
      +--ro uuid          yang:uuid
      .....
    +--ro shelves* [uuid]
      +--ro uuid          yang:uuid
      .....
    +--ro chassis-ref
      +--ro ne-ref?       leafref
      +--ro component-ref? leafref
  +--ro network-elements
    +--ro network-element* [uuid]
      +--ro uuid          yang:uuid
      .....
    +--ro components
      +--ro component* [uuid]
        +--ro uuid          yang:uuid
        .....

```

The YANG data model for network inventory follows the same approach of [RFC8348] and reports the network inventory as a list of components of different types (e.g., chassis, module, port).

```

+--ro components
  +--ro component* [uuid]
    +--ro uuid          yang:uuid
    +--ro name?         string
    +--ro description?  string
    +--ro class?        identityref
    +--ro parent-rel-pos? int32
    +--ro children* [child-ref]
      | +--ro child-ref  -> ../../../../uuid
    +--ro parent
      +--ro parent-ref? -> ../../../../uuid

```

Note: review in future versions of this document whether the component list should be under the network-inventory instead of under the network-element container

However, considering there are some special scenarios, the relationship between the rack and network elements is not 1 to 1 nor 1 to n. The network element cannot be the direct parent node of the rack. So there should be n to m relationship between racks and network elements. And the shelves in the rack should have some reference information to the component.

Note that in [RFC8345], topology and inventory are two subsets of network information. However, considering the complexity of the existing topology models and to have a better extension capability, we define a separate root for the inventory model. We will consider some other ways to do some associations between the topology model and inventory model in the future.

Note: review in future versions of this document whether network inventory should be defined as an augmentation of the network model defined in [RFC8345] instead of under a new network-inventory root.

The proposed YANG data model has been analysed to cover the requirements and use cases for Optical Network Inventory.

Further analysis of requirements and use cases is needed to extend the applicability of this YANG data model to other types of networks (IP and microwave) and to identify which aspects are generic and which aspects are technology-specific for optical networks.

3. Optical Network Inventory Tree Diagram

Figure 2 below shows the tree diagram of the YANG data model defined in module ietf-network-inventory.yang (Section 4).

```

module: ietf-network-inventory
+--ro network-inventory
  +--ro equipment-rooms
    +--ro equipment-room* [uuid]
      +--ro uuid          yang:uuid
      +--ro name?         string
      +--ro location?     string
      +--ro rack* [uuid]
        +--ro uuid          yang:uuid
        +--ro name?         string
        +--ro row-number?   uint32
        +--ro rack-number?  uint32
        +--ro shelves* [uuid]
          +--ro uuid          yang:uuid
          +--ro name?         string
          +--ro shelf-number? uint8
          +--ro chassis-ref
            +--ro ne-ref?      leafref
            +--ro component-ref? leafref
      +--ro network-elements
        +--ro network-element* [uuid]
          +--ro uuid          yang:uuid
          +--ro name?         string
          +--ro components
            +--ro component* [uuid]
              +--ro uuid          yang:uuid
              +--ro name?         string
              +--ro description?   string
              +--ro class?         identityref
              +--ro parent-rel-pos? int32
              +--ro children* [child-ref]
              | +--ro child-ref    -> ../../../../uuid
              +--ro parent
                +--ro parent-ref?  -> ../../../../uuid

```

Figure 2: Network inventory tree diagram

4. YANG Model for Optical Network Inventory

```

<CODE BEGINS> file "ietf-network-inventory@2021-10-25.yang"
module ietf-network-inventory {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-inventory";
  prefix ni;

  import ietf-yang-types {
    prefix yang;
    reference

```

```
    "RFC6991: Common YANG Data Types.";
}

import iana-hardware {
  prefix ianahw;
  reference
    "RFC 8348: A YANG Data Model for Hardware Management.";
}

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

  Editor:   Chaode Yu
            <yuchaode@huawei.com>

  Editor:   Italo Busi
            <italo.busi@huawei.com>

  Editor:   Aihua Guo
            <aihuaguo.ietf@gmail.com>

  Editor:   Sergio Belotti
            <sergio.belotti@nokia.com>

  Editor:   Jean-Francois Bouquier
            <jeff.bouquier@vodafone.com>

  Editor:   Fabio Peruzzini
            <fabio.peruzzini@telecomitalia.it>";

description
  "This module defines a model for retrieving network inventory.

  The model fully conforms to the Network Management
  Datastore Architecture (NMDA).

  Copyright (c) 2021 IETF Trust and the persons
  identified as authors of the code.  All rights reserved.

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

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

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.";

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

```
revision 2021-10-25 {
  description
    "Initial revision.";
  reference
    "draft-yg3bp-ccamp-optical-inventory-yang-00: A YANG Data
    Model for Optical Network Inventory.";
}
```

```
container network-inventory {
  config false;
  description
    "The top-level container for the network inventory
    information.";
  uses equipment-rooms-grouping;
  uses network-elements-grouping;
}
```

```
grouping common-entity-attributes {
  description
    "A set of attributes which are common to all the entities
    (e.g., component, equipment room) defined in this module.";
  leaf uuid {
    type yang:uuid;
    description
      "Uniquely identifies an entity (e.g., component).";
  }
  leaf name {
    type string;
    description
      "A name for an entity (e.g., component), as specified by
      a network manager, that provides a non-volatile 'handle'
      for the entity and that can be modified anytime during the
      entity lifetime.
```

```
        If no configured value exists, the server MAY set the value
        of this node to a locally unique value in the operational
        state.";
    }
}
grouping network-elements-grouping {
    description
        "The attributes of the network elements.";
    container network-elements {
        description
            "The container for the list of network elements.";
        list network-element {
            key uuid;
            description
                "The list of network elements within the network.";
            uses common-entity-attributes;
            uses components-grouping;
        }
    }
}

grouping equipment-rooms-grouping {
    description
        "The attributes of the equipment rooms.";
    container equipment-rooms {
        description
            "The container for the list of equipment rooms.";
        list equipment-room {
            key uuid;
            description
                "The list of equipment rooms within the network.";
            uses common-entity-attributes;
            leaf location {
                type string;
                description
                    "compared with the location information of the other
                    inventory objects, a GIS address is preferred for
                    equipment room";
            }
        }
        list rack {
            key uuid;
            description
                "The list of racks within an equipment room.";
            uses common-entity-attributes;
            leaf row-number {
                type uint32;
                description
                    "Identifies the row within the equipment room where
```



```
}

groupings components-grouping {
  description
    "The attributes of the hardware components.";
  container components {
    description
      "The container for the list of components.";
    list component {
      key uuid;
      description
        "The list of components within a network element.";
      uses common-entity-attributes;
      leaf description {
        type string;
        description
          "A textual description of the component.";
        reference
          "RFC 8348: A YANG Data Model for Hardware Management.";
      }
      leaf class {
        type identityref {
          base ianahw:hardware-class;
        }
        description
          "An indication of the general hardware type of the
           component.";
        reference
          "RFC 8348: A YANG Data Model for Hardware Management.";
      }
      leaf parent-rel-pos {
        type int32 {
          range "0 .. 2147483647";
        }
        description
          "An indication of the relative position of this child
           component among all its sibling components. Sibling
           components are defined as components that:

              o share the same value of the 'parent' node and

              o share a common base identity for the 'class' node.";
        reference
          "RFC 8348: A YANG Data Model for Hardware Management.";
      }
      list children {
        key child-ref;
        description
```

```

    "The child components that are physically contained by
    this component.";

    leaf child-ref {
      type leafref {
        path "../..../..../ni:uuid";
      }
      description
        "The reference to the child component.";
    }
  }
  container parent {
    description
      "The parent component that physically contains this
      component.

      If this container is not instantiated, it indicates
      that this component is not contained in any other
      component.

      In the event that a physical component is contained by
      more than one physical component (e.g., double-wide
      modules), this container contains the data of one of
      these components. An implementation MUST use the same
      component every time this container is instantiated.";
    leaf parent-ref {
      type leafref {
        path "../..../..../ni:uuid";
      }
      description
        "The reference to the parent component.";
    }
  }
}
}
}
}
}
<CODE ENDS>

```

Figure 3: Network inventory YANG module

5. Manageability Considerations

<Add any manageability considerations>

6. Security Considerations

<Add any security considerations>

7. IANA Considerations

<Add any IANA considerations>

8. References

8.1. Normative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7446] Lee, Y., Ed., Bernstein, G., Ed., Li, D., and W. Imajuku, "Routing and Wavelength Assignment Information Model for Wavelength Switched Optical Networks", RFC 7446, DOI 10.17487/RFC7446, February 2015, <<https://www.rfc-editor.org/info/rfc7446>>.
- [RFC7581] Bernstein, G., Ed., Lee, Y., Ed., Li, D., Imajuku, W., and J. Han, "Routing and Wavelength Assignment Information Encoding for Wavelength Switched Optical Networks", RFC 7581, DOI 10.17487/RFC7581, June 2015, <<https://www.rfc-editor.org/info/rfc7581>>.
- [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>>.
- [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>>.

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <<https://www.rfc-editor.org/info/rfc8348>>.

[TMF-MTOSI] TM Forum (TMF), "TMF MTOSI 4.0 Equipment Model", TMF SD2-20_EquipmentModel , 2008, <<https://www.tmforum.org/resources/suite/mtosi-4-0/>>.

8.2. Informative References

[I-D.ietf-teas-actn-poi-applicability] Peruzzini, F., Bouquier, J., Busi, I., King, D., and D. Ceccarelli, "Applicability of Abstraction and Control of Traffic Engineered Networks (ACTN) to Packet Optical Integration (POI)", Work in Progress, Internet-Draft, draft-ietf-teas-actn-poi-applicability-03, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-actn-poi-applicability-03.txt>>.

[ONF_TR-547] Open Networking Foundation (ONF), "TAPI v2.1.3 Reference Implementation Agreement", ONF TR-547 TAPI RIA v1.0 , July 2020, <<https://opennetworking.org/wp-content/uploads/2020/08/TR-547-TAPI-v2.1.3-Reference-Implementation-Agreement-1.pdf>>.

[RFC8345] Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A YANG Data Model for Network Topologies", RFC 8345, DOI 10.17487/RFC8345, March 2018, <<https://www.rfc-editor.org/info/rfc8345>>.

Acknowledgments

The authors of this document would like to thank the authors of [I-D.ietf-teas-actn-poi-applicability] for having identified the gap and requirements to trigger this work.

This document was prepared using kramdown.

Authors' Addresses

Chaode Yu
Huawei Technologies

Email: yuchaode@huawei.com

Italo Busi
Huawei Technologies

Email: italo.busi@huawei.com

Aihua Guo
Futurewei Technologies

Email: aihuaguo.ietf@gmail.com

Sergio Belotti
Nokia

Email: sergio.belotti@nokia.com

Jean-Francois Bouquier
Vodafone

Email: jeff.bouquier@vodafone.com

Fabio Peruzzini
TIM

Email: fabio.peruzzini@telecomitalia.it

Oscar Gonzalez de Dios
Telefonica

Email: oscar.gonzalezdedios@telefonica.com

Victor Lopez
Nokia

Email: victor.lopez@nokia.com