Node Tags in YANG Modules
draft-ietf-netmod-node-tags-08

Abstract

This document defines a method to tag nodes that are associated with operation and management data in YANG modules. This method for tagging YANG nodes is meant to be used for classifying either data nodes or instances of data nodes from different YANG modules and identifying their characteristic data. Tags may be registered as well as assigned during the definition of the module, assigned by implementations, or dynamically defined and set by users.

This document also provides guidance to future YANG data model writers; as such, this document updates RFC 8407.

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 24 December 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
Table of Contents

1. Introduction ................................................. 3
2. Terminology ................................................. 4
3. Sample Use Cases for Node Tags ............................. 5
4. Node Tag Values ............................................. 6
   4.1. IETF Tags .................................................. 6
   4.2. Vendor Tags ............................................... 6
   4.3. User Tags .................................................. 6
   4.4. Reserved Tags ............................................. 7
5. Node Tag Management ......................................... 7
   5.1. Module Design Tagging ................................... 7
   5.2. Implementation Tagging .................................. 7
   5.3. User Tagging .............................................. 7
6. Node Tags Module Structure .................................. 7
   6.1. Node Tags Module Tree ................................... 7
7. Node Tags YANG Module ....................................... 8
8. Guidelines to Model Writers ................................ 12
   8.1. Define Standard Tags .................................... 12
9. IANA Considerations ......................................... 13
   9.1. YANG Data Node Tag Prefixes Registry .................. 13
   9.2. IETF YANG Data Node Tags Registry ...................... 14
   9.3. Updates to the IETF XML Registry ....................... 15
   9.4. Updates to the YANG Module Names Registry .......... 15
10. Security Considerations ...................................... 16
11. Acknowledgements ........................................... 16
12. Contributors ................................................ 17
13. References .................................................... 17
   13.1. Normative References ................................... 17
   13.2. Informative References ................................ 18
Appendix A. Example: Additional Auxiliary Data Property
   Information ................................................... 19
Appendix B. Instance Level Tunnel Tagging Example ............ 20
Appendix C. NETCONF Example .................................... 22
Appendix D. Non-NMDA State Module .............................. 23
Appendix E. Targeted Data Fetching Example .................... 27
Appendix F. Changes between Revisions ........................... 29
Authors’ Addresses .............................................. 31
1. Introduction

The use of tags for classification and organization purposes is fairly ubiquitous, not only within IETF protocols, but globally in the Internet (e.g., "#hashtags"). For the specific case of YANG data models, a module tag is defined as a string that is associated with a module name at the module level [RFC8819].

Many data models have been specified by various Standards Developing Organizations (SDOs) and the Open Source community, and it is likely that many more will be specified. These models cover many of the networking protocols and techniques. However, data nodes defined by these technology-specific data models might represent only a portion of fault, configuration, accounting, performance, and security (FCAPS) management information ([FCAPS]) at different levels and network locations, but also categorized in various different ways. Furthermore, there is no consistent classification criteria or representations for a specific service, feature, or data source.

This document defines tags for both nodes in the schema tree and instance nodes in the data tree and shows how they can be associated with nodes within a YANG module, which:

* Provide dictionary meaning for specific targeted data nodes;

* Indicate a relationship between data nodes within the same YANG module or from different YANG modules;

* Identify auxiliary data properties related to data nodes;

* Identify key performance metric related data nodes and the absolute XPath expression identifying the element path to the nodes.

To that aim, this document defines a YANG module [RFC7950] that augments the YANG Module Tags ([RFC8819]) to provide a list of node entries to add or remove node tags as well as to view the set of node tags associated with specific data nodes or instance of data nodes within YANG modules. This new module is: "ietf-node-tags" (Section 7).

Typically, NETCONF clients can discover node tags supported by a NETCONF server by means of the <get-data> operation on the operational datastore (Section 3.1 of [RFC8526]) via the "ietf-node-tags" module. Alternatively, <get-schema> operation can be used to retrieve tags for nodes in the schema tree in any data module. These node tags can be used by a NETCONF [RFC6241] or RESTCONF [RFC8040] client to classify either data nodes or instance of these data nodes.
from different YANG modules and identify characteristic data and
associated path to the nodes or node instances. Therefore, the
NETCONF/RESTCONF client can query specific configuration or
operational state on a server corresponding to characteristic data.

Similar to YANG module tags defined in [RFC8819], these node tags
(e.g., tags for node in the schema node) may be registered or
assigned during the module definition, assigned (e.g., tags for nodes
in the data tree) by implementations, or dynamically defined and set
by users. The contents of node tags from the operational state view
are constructed using the following steps:

1. System tags (i.e., tags of "system" origin) that assigned during
   the module definition time are added;

2. User-configured tags (i.e., tags of "intended" origin) that
   dynamically defined and set by users at runtime;

3. Any tag that is equal to a masked-tag is removed.

This document defines an extension statement to indicate tags for
data nodes. YANG metadata annotations are also defined in [RFC7952]
as a YANG extension. The value of YANG metadata annotations is
attached to a given data node instance and decided and assigned by
the server and sent to the client (e.g., the origin value indicates
to the client the origin of a particular data node instance) while
tags for data node in the schema tree defined in Section 7 are
retrieved centrally via the "ietf-node-tags" module and can be
dynamically set by the client.

This document also defines an IANA registry for tag prefixes and a
set of globally assigned tags (Section 9).

Section 8 provides guidelines for authors of YANG data models. This
document updates [RFC8407].

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in BCP
14 [RFC2119][RFC8174] when, and only when, they appear in all
capitals, as shown here.
The following terms are defined in [RFC7950] and are not redefined here:

* Data Node
* Data Tree
* Schema Tree

This document defines the following term:

Node Tag: Tag for YANG nodes used for classifying either data nodes or instances of data nodes from different YANG modules and identifying their characteristic data.

The meanings of the symbols in tree diagrams are defined in [RFC8340].

3. Sample Use Cases for Node Tags

The following lists a set of use cases to illustrate the use of node tags. This section does not intend to be exhaustive.

An example of the use of tags is to search discrete categories of YANG nodes that are scattered across the same or different YANG modules supported by a device. For example, if instances of these nodes in YANG modules are adequately tagged and set by a first client ("client A") via the "ietf-node-tags" module (Section 7) and retrieved by another client ("client B") from the operational datastore, then "client B" can obtain the path to the tagged nodes and subscribe only to network performance related data node instances in the operational datastore supported by a device.

"Client B" can also subscribe to updates from the operational datastore using the "ietf-node-tags" module. Any tag changes in the updates will then resynchronize to the "client B".

Also, tag classification is useful for users searching data nodes repositories. A query restricted to the "ietf:counter" data node tag in the "ietf-node-tags" module can be used to return only the YANG nodes that are associated with the counter. Without tags, a user would need to know the name of all the IETF YANG data nodes or instances of data nodes in different YANG modules.

Future management protocol extensions could allow for filtering queries of configuration or operational state on a server based on tags (for example, return all operational state related to system management).
4. Node Tag Values

All node tags (except in some cases of user tags as described in Section 4.3) begin with a prefix indicating who owns their definition. An IANA registry (Section 9.1) is used to register node tag prefixes. Initially, three prefixes are defined.

No further structure is imposed by this document on the value following the registered prefix, and the value can contain any YANG type 'string' characters except carriage returns, newlines, tabs, and spaces.

Except for the conflict-avoiding prefix, this document is purposefully not specifying any structure on (i.e., restricting) the tag values. The intent is to avoid arbitrarily restricting the values that designers, implementers, and users can use. As a result of this choice, designers, implementers, and users are free to add or not add any structure they may require to their own tag values.

4.1. IETF Tags

An IETF tag is a node tag that has the prefix "ietf:"

All IETF node tags are registered with IANA in the registry defined in Section 9.2.

4.2. Vendor Tags

A vendor tag is a tag that has the prefix "vendor:"

These tags are defined by the vendor that implements the module, and are not registered with IANA. However, it is RECOMMENDED that the vendor includes extra identification in the tag to avoid collisions, such as using the enterprise or organization name following the "vendor:" prefix (e.g., vendor:entno:vendor-defined-classifier).

4.3. User Tags

User tags are defined by a user/administrator and are not registered by IANA.

Any tag with the prefix "user:" is a user tag. Furthermore, any tag that does not contain a colon (":", i.e., has no prefix) is also a user tag. Users are not required to use the "user:" prefix; however, doing so is RECOMMENDED.
4.4. Reserved Tags

Section 9.1 describes the IANA registry of tag prefixes. Any prefix not included in that registry is reserved for future use, but tags starting with such a prefix are still valid tags.

5. Node Tag Management

Tags may be associated with a data node within a YANG module in a number of ways. Typically, tags may be defined and associated at the module design time, at implementation time without the need of a live server, or via user administrative control. As the main consumers of node tags are users, users may also remove any tag from a live server, no matter how the tag became associated with a data node within a YANG module.

5.1. Module Design Tagging

A data node definition MAY indicate a set of node tags to be added by a module’s implementer. These design time tags are indicated using ‘node-tag’ extension statement.

If the data node is defined in an IETF Standards Track document, node tags MUST be IETF Tags (Section 4.1). Thus, new data nodes can drive the addition of new IETF tags to the IANA registry defined in Section 9.2, and the IANA registry can serve as a check against duplication.

5.2. Implementation Tagging

An implementation MAY include additional tags associated with data nodes within a YANG module. These tags SHOULD be IETF ((i.e., registered) ) or vendor tags.

5.3. User Tagging

Node tags of any kind, with or without a prefix, can be assigned and removed by the user from a server using normal configuration mechanisms. In order to remove a node tag from the operational datastore, the user adds a matching “masked-tag” entry for a given node within the ’ietf-node-tags’ module.

6. Node Tags Module Structure

6.1. Node Tags Module Tree

The tree associated with the "ietf-node-tags" module is as follows:
module: ietf-node-tags  
augment /tags:module-tags/tags:module: 
  +--rw node-tags 
  |  +--rw node* [id] 
  |     |  +--rw id  nacm:node-instance-identifier 
  |     |  +--rw tags* [tag] 
  |     |     |  +--rw tag  tags:tag 
  |     |  +--rw type?  identityref 
  +--rw masked-tag*  tags:tag

Figure 1: YANG Module Node Tags Tree Diagram

7. Node Tags YANG Module

The "ietf-node-tags" module imports types from [RFC8819] and
[RFC8341].

<CODE BEGINS> file "ietf-node-tags@2022-02-04.yang"
module ietf-node-tags { 
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-node-tags";
  prefix ntags;

  import ietf-netconf-acm {
    prefix nacm;
    reference
    "RFC 8341: Network Configuration Access Control
     Model";
  }

  import ietf-module-tags {
    prefix tags;
    reference
    "RFC 8819: YANG Module Tags ";
  }

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

  Editor: Qin Wu
  <mailto:bill.wu@huawei.com>

  Editor: Benoit Claise
  <mailto:benoit.claise@huawei.com>

  Editor: Peng Liu

This module describes a mechanism associating tags with YANG node within YANG modules. Tags may be IANA assigned or privately defined.

Copyright (c) 2022 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 Revised 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 (https://datatracker.ietf.org/html/rfcXXXX); see the RFC itself for full legal notices.

// RFC Ed.: update the date below with the date of RFC publication and RFC number and remove this note.
revision 2022-02-04 {
    description
    "Initial revision.";
    reference
    "RFC XXXX: Node Tags in YANG Modules";
}
identity node-tag-type {
    description
    "Base identity for node tag type.";
}
identity metric {
    base node-tag-type;
    description
    "Identity for metric tag type.";
}
identity delay {
    base node-tag-type;
    description
"Identity for delay metric tag type."
}
identity jitter {
    base node-tag-type;
    description
        "Identity for jitter metric tag type."
}
identity loss {
    base node-tag-type;
    description
        "Identity for loss metric tag type."
}
identity counter {
    base node-tag-type;
    description
        "Identity for counter metric tag type."
}
identity summary {
    base node-tag-type;
    description
        "Identity for summary metric tag type."
}
identity gauge {
    base node-tag-type;
    description
        "Identity for gauge metric tag type."
}
identity unknown {
    base node-tag-type;
    description
        "Identity for unkown metric tag type."
}
identity agg {
    base node-tag-type;
    description
        "Identity for aggregated metric tag type."
}
extension node-tag {
    argument tag;
    description
        "The argument 'tag' is of type 'tag'. This extension statement
        is used by module authors to indicate node tags that should
        be added automatically by the system. As such, the origin of
        the value for the pre-defined tags should be set to 'system'."
}

augment "/tags:module-tags/tags:module" {
    description

"Augment the Module Tags module with node tag attributes."

container node-tags {
  description "Contains the list of nodes or node instances and their associated node tags.";
  list node {
    key "id";
    description "Includes a list of nodes and their associated node tags.";
    leaf id {
      type nacm:node-instance-identifier;
      description "The YANG data node name or data node instance name.";
    }
    list tags {
      key "tag";
      description "Lists the tags associated with the node within the YANG module."
      See the IANA 'YANG node Tag Prefixes' registry for reserved prefixes and the IANA 'IETF YANG Data Node Tags' registry for IETF tags.

      The 'operational' state view of this list is constructed using the following steps:

      1) System tags (i.e., tags of 'system' origin) are added.
      2) User configured tags (i.e., tags of 'intended' origin) are added.
      3) Any tag that is equal to a masked-tag is removed.";
      reference "RFC XXXX: node Tags in YANG Data Modules, Section 9";
    }

    leaf tag {
      type tags:tag;
      description "Node tag corresponding to type of node tag.";
    }

    leaf type {
      type identityref {
        base node-tag-type;
      }
      description "Type of node tag.";
    }
  }
}
leaf-list masked-tag {
  type tags:tag;
  description
    "The list of tags that should not be associated with the
    node within the YANG module. The user can remove
    (mask) tags from the operational state datastore by
    adding them to this list. It is not an error to add tags
    to this list that are not associated with the data
    node within YANG module, but they have no operational
    effect."
};
}
}
}
}
</CODE ENDS>

8. Guidelines to Model Writers

This section updates [RFC8407] by providing text that may be regarded
as a new subsection to Section 4 of that document. It does not
change anything already present in [RFC8407].

8.1. Define Standard Tags

A module MAY indicate, using node tag extension statements, a set of
node tags that are to be automatically associated with node within
the module (i.e., not added through configuration).

module example-module-A {
  //...
  import ietf-node-tags { prefix ntags; }

  container top {
    list X {
      leaf foo {
        ntags:node-tag "ietf:summary";
      }
      leaf bar {
        ntags:node-tag "ietf:loss";
      }
    }
  }
  // ...
}
The module writer can use existing standard node tags, or use new node tags defined in the data node definition, as appropriate. For IETF standardized modules, new node tags MUST be assigned in the IANA registry defined in Section 9.2.

9. IANA Considerations

9.1. YANG Data Node Tag Prefixes Registry

This document requests IANA to create "YANG node Tag Prefixes" subregistry in "YANG node Tag" registry.

Prefix entries in this registry should be short strings consisting of lowercase ASCII alpha-numeric characters and a final ":" character.

The allocation policy for this registry is Specification Required [RFC8126]. The Reference and Assignee values should be sufficient to identify and contact the organization that has been allocated the prefix. There is no specific guidance for the Designated Expert and there is a presumption that a code point should be granted unless there is a compelling reason to the contrary.

The initial values for this registry are as follows:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
<th>Reference</th>
<th>Assignee</th>
</tr>
</thead>
<tbody>
<tr>
<td>ietf:</td>
<td>IETF Tags allocated in the IANA YANG node Tags registry</td>
<td>[This document]</td>
<td>IETF</td>
</tr>
<tr>
<td>vendor:</td>
<td>Non-registered tags allocated by the module’s implementer.</td>
<td>[This document]</td>
<td>IETF</td>
</tr>
<tr>
<td>user:</td>
<td>Non-registered tags allocated by and for the user.</td>
<td>[This document]</td>
<td>IETF</td>
</tr>
</tbody>
</table>

Other standards organizations (SDOs) wishing to allocate their own set of tags should request the allocation of a prefix from this registry.
9.2. IETF YANG Data Node Tags Registry

This document requests IANA to create "IETF Node Tags" subregistry in "YANG node Tag" registry. This subregistry appears below "YANG node Tag Prefixes" registry.

This subregistry allocates tags that have the registered prefix "ietf:". New values should be well considered and not achievable through a combination of already existing IETF tags.

The allocation policy for this subregistry is IETF Review [RFC8126]. The Designated Expert is expected to verify that IANA assigned tags conform to Net-Unicode as defined in [RFC5198], and shall not need normalization.

The initial values for this subregistry are as follows:

<table>
<thead>
<tr>
<th>Node Tag</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ietf:metric</td>
<td>Represent metric data (e.g., ifstatistics)</td>
<td>[This document]</td>
</tr>
<tr>
<td></td>
<td>associated with specific node (e.g., interfaces)</td>
<td></td>
</tr>
<tr>
<td>ietf:delay</td>
<td>Represents the delay metric data associated with specific node.</td>
<td>[This document]</td>
</tr>
<tr>
<td>ietf:jitter</td>
<td>Represents the jitter metric data associated with specific node.</td>
<td>[This document]</td>
</tr>
<tr>
<td>ietf:loss</td>
<td>Represents the loss metric data associated with specific node.</td>
<td>[This document]</td>
</tr>
<tr>
<td>ietf:counter</td>
<td>Represents any metric value associated with specific node that monotonically increases over time, starting from zero.</td>
<td>[This document]</td>
</tr>
<tr>
<td>ietf:gauge</td>
<td>Represents current measurements associated with specific node</td>
<td>[This document]</td>
</tr>
</tbody>
</table>
that may increase, decrease or stay constant.

<table>
<thead>
<tr>
<th>ietf:summary</th>
<th>Represents the metric value associated with specific node that measures distributions of discrete events without knowing predefined range.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ietf:unknown</td>
<td>Represents the metric value associated with specific node that can not determine the type of metric.</td>
</tr>
<tr>
<td>ietf:agg</td>
<td>Relates to aggregated metric value associated with specific node (i.e., aggregated statistics)</td>
</tr>
</tbody>
</table>

Figure 4: Table 2

A data node can contain one or multiple node tags. Data node to be tagged with the initial value in Table 2 can be one of 'container', 'leaf-list', 'list', or 'leaf' data node. All tag values described in Table 2 can be inherited down the containment hierarchy if Data nodes tagged with those tag values is one of 'container', 'leaf-list', 'list'.

9.3. Updates to the IETF XML Registry

This document registers the following namespace URI in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

9.4. Updates to the YANG Module Names Registry

This document registers the following YANG module in the YANG Module Names registry [RFC6020] within the "YANG Parameters" registry:
10. Security Considerations

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

The 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, e.g., the presence of tags may reveal information about the way in which data nodes or node instances are used and therefore providing access to private information or revealing an attack vector should be restricted. Note that appropriate privilege and security levels need to be applied to the addition and removal of user tags to ensure that a user receives the correct data.

This document adds the ability to associate node tag with data nodes or instances of data nodes within the YANG modules. This document does not define any actions based on these associations, and none are yet defined, and therefore it does not by itself introduce any new security considerations.

Users of the node tag meta-data may define various actions to be taken based on the node tag meta-data. These actions and their definitions are outside the scope of this document. Users will need to consider the security implications of any actions they choose to define, including the potential for a tag to get 'masked' by another user.

11. Acknowledgements

The authors would like to thank Ran Tao for his major contributions to the initial modeling and use cases.
The authors would also like to acknowledge the comments and suggestions received from Juergen Schoenwaelder, Andy Bierman, Lou Berger, Jaehoon Paul Jeong, Wei Wang, Yuan Zhang, Ander Liu, YingZhen Qu, Boyuan Yan, Adrian Farrel, and Mahesh Jethanandani.

12. Contributors

Liang Geng  
Individual  
32 Xuanwumen West St, Xicheng District  
Beijing 10053

13. References

13.1. Normative References


13.2. Informative References


Appendix A. Example: Additional Auxiliary Data Property Information

This section gives an example of how Auxiliary Data Property Module could be defined. It demonstrates how auxiliary data property configuration parameters can be conditionally augmented to the generic node list. The example is not intended as a complete module for Auxiliary Data Property configuration.
module ex-auxiliary-data-property {
  yang-version 1.1;
  namespace "http://example.com/auxiliary-data-property";
  prefix "dp";

  import ietf-module-tags {
    prefix tags;
  }
  import ietf-node-tags {
    prefix ntags;
  }

  identity critical {
    base ntags:node-tag-type;
    description "Identity for critical node tag type.";
  }

  augment "/tags:module-tags/tags:module/ntags:node-tags/ntags:" + "node/ntags:tags"
    when 'derived-from-or-self(ntags:type, "dp:critical")';
  description "Extend ietf-node-tags module for auxiliary data property.";
  leaf value {
    type string;
    description "The auxiliary information corresponding to data node instance tagged with 'critical' node tag type.";
  }

  // other auxiliary data property config params, etc.
}

Appendix B. Instance Level Tunnel Tagging Example

In the example shown in the following figure, the 'tunnel-svc' data node is a list node defined in a 'example-tunnel-pm' module and has 7 child nodes: 'name', 'create-time', 'modified-time', 'average-latency', 'packet-loss', 'min-latency', 'max-latency' leaf node. In these child nodes, the 'name' leaf node is the key leaf for the 'tunnel-svc' list. Following is the tree diagram [RFC8340] for the "example-tunnel-pm" module:
---rw tunnel-svc* [name]
  +--rw name                    string
  +--ro create-time             yang:date-and-time
  +--ro modified-time          yang:date-and-time
  +--ro average-latency        yang:gauge64
  +--ro packet-loss            yang:counter64
  +--ro min-latency            yang:gauge64
  +--ro max-latency            yang:gauge64

To help identify specific data for a customer, users tags on specific instances of the data nodes are created as follows:

    <rpc message-id="103"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <datastore>ds:running</datastore>
        <config>
          <module-tag>
            <module>
              <name>example-tunnel-pm</name>
              <node-tags xmlns="urn:ietf:params:xml:ns:yang:ietf-node-tags">
                <node>
                  <id>/tp:tunnel-svc[name='foo']/tp:packet-loss</id>
                  <tags>
                    <tag>user:customer1_example_com</tag>
                  </tags>
                </node>
                <node>
                  <id>/tp:tunnel-svc[name='bar']/tp:modified-time</id>
                  <tags>
                    <tag>user:customer2_example_com</tag>
                  </tags>
                </node>
              </node-tags>
            </module>
          </module-tag>
        </config>
      </edit-data>
    </rpc>
Note that the ‘ietf:critical’ tag is additional new tag value that needs to be allocated from "IETF Node Tags" subregistry in Section 9.2.

Appendix C. NETCONF Example

The following is a NETCONF example result from a query of node tags list. For the sake of brevity only a few module and associated data node results are provided. The example uses the folding defined in [RFC8792].
Figure 5: Example NETCONF Query Output

Appendix D. Non-NMDA State Module

As per [RFC8407], the following is a non-NMDA module to support viewing the operational state for non-NMDA compliant servers.
<CODE BEGINS> file "ietf-node-tags-state@2022-02-03.yang"
module ietf-node-tags-state {
    yang-version 1.1;
    namespace
        "urn:ietf:params:xml:ns:yang:ietf-node-tags-state";
    prefix ntags-s;

    import ietf-netconf-acm {
        prefix nacm;
        reference
            "RFC 8341: Network Configuration Access Control Model";
    }

    import ietf-module-tags {
        prefix tags;
    }

    import ietf-module-tags-state {
        prefix tags-s;
        reference
            "RFC 8819: YANG Module Tags ";
    }

    organization
        "IETF NetMod Working Group (NetMod)";

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

        Editor: Qin Wu
            <mailto:bill.wu@huawei.com>

        Editor: Benoit Claise
            <mailto:benoit.claise@huawei.com>

        Editor: Peng Liu
            <mailto:liupengyjy@chinamobile.com>

        Editor: Zongpeng Du
            <mailto:duzongpeng@chinamobile.com>

        Editor: Mohamed Boucadair
            <mailto:mohamed.boucadair@orange.com>";
            // RFC Ed.: replace XXXX with actual RFC number and
            // remove this note.
    description
        "This module describes a mechanism associating data node
         tags with YANG data node within YANG modules. Tags may be
         IANA assigned or privately defined."
Copyright (c) 2022 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 (https://datatracker.ietf.org/html/rfcXXXX); see the RFC itself for full legal notices.

// RFC Ed.: update the date below with the date of RFC publication and RFC number and remove this note.
revision 2022-02-04 {
    description
        "Initial revision.";
    reference
        "RFC XXXX: Node Tags in YANG Data Modules";
}

identity node-tag-type {
    description
        "Base identity for node tag type.";
}

augment "tags-s:module-tags-state/tags-s:module" {
    description
        "Augments the Module Tags module with node tag attributes.";
    container node-tags {
        config false;
        status deprecated;
        description
            "Contains the list of data nodes and their associated self describing tags.";
    list node {
        key "id";
        status deprecated;
        description
            "Lists the data nodes and their associated self describing tags.";
        leaf id {
            type nacm:node-instance-identifier;
            mandatory true;
            status deprecated;
            description
"The YANG data node name.");
}
list tags {
key "tag";
status deprecated;
description
"Lists the tags associated with the data node within the YANG module.

See the IANA 'YANG node Tag Prefixes' registry for reserved prefixes and the IANA 'IETF YANG Data Node Tags' registry for IETF tags.

The 'operational' state view of this list is constructed using the following steps:

1) System tags (i.e., tags of 'system' origin) are added.
2) User configured tags (i.e., tags of 'intended' origin) are added.
3) Any tag that is equal to a masked-tag is removed.";
reference
"RFC XXXX: Node Tags in YANG Data Modules, Section 9";

leaf tag {
type tags:tag;
status deprecated;
description
"Node tag corresponding to type of node tag.";
}
leaf type {
type identityref {
base node-tag-type;
}
status deprecated;
description "type of the node tag.";
}
}
leaf-list masked-tag {
type tags:tag;
status deprecated;
description
"The list of tags that should not be associated with the data node within the YANG module. The user can remove (mask) tags from the operational state datastore by adding them to this list. It is not an error to add tags to this list that are not associated with the data node within YANG module, but they have no
Appendix E. Targeted Data Fetching Example

The following provides tagged data node fetching example. The subscription "id" values of 22 used below is just an example. In production, the actual values of "id" might not be small integers.

```
+-----------+                        +-----------+
| Subscriber|                        | Publisher |
+-----------+                        +-----------+

| Node Tagging Fetching |
| (id, node-tag = metric) |
|<------------------------|
| establish-subscription |
|+------------------------+
| RPC Reply: OK, id = 22  |
|<------------------------|
| Notification Message (for 22) |
|<------------------------|
```

The subscriber can query node tag list from operational datastore in the network device using "ietf-node-tags" module defined in this document and fetch tagged data node instances and associated data path to the datastore node. The node tag information instruct the receiver to subscribe tagged data node (e.g., performance metric data nodes) using standard subscribed notification mechanism [RFC8639].
With node tag information returned, e.g., in the 'get-data' operation, the subscriber identifies tagged data node and associated data path to the datastore node and sends a standard establish-subscription RPC [RFC8639] to subscribe tagged data nodes that are interests to the client application from the publisher. The publisher returns specific data node types of operational state (e.g., in-errors statistics data) subscribed by the client as follows:
Appendix F. Changes between Revisions

Editorial Note (To be removed by RFC Editor)

v07 - v08

* Make objective clearly, cover tags for both nodes in the schema tree and nodes in the data tree.

* Document clearly which tags can be cached and how applications are supposed to resynchronize and pull in any update in section 3.

* Clarify Instance level tag is not used to guide retrieval operations in section 3.

* Distinguish Instance level tag from Metadata annotation in the introduction section.

* Distinguish Schema Level tag from Instance level tag in the introduction section and section 3.

* Schema Level tag used in xpath query has been clarified in section 3.

* Other editorial changes.
v06 - v07

* Update use case in section 3 to remove object and subobject concept and massive related words.

* Change the title into Node Tags in YANG Modules.

* Update Model Tag design in section 5.1 based on Balazs’s comments.

* Add Instance level tunnel tagging example in the Appendix.

* Add 'type' parameter in the base model and add one more model extension example in the Appendix.

* Consolidate opm-tag extension, metric-type extension and multi-source-tag extension into one generic yang extension.

* Remove object tag and property tag.

* Other Appendix Updates.

v05 - v06

* Additional Editorial changes;

* Use the folding defined in [RFC8792].

v04 - v05

* Add user tag formating clarification;

* Provide guidance to the Designated Expert for evaluation of YANG node Tag registry and YANG node Tag prefix registry.

* Update the figure 1 and figure 2 with additional tags.

* Security section enhancement for user tag managment.

* Change data node name into name in the module.

* Other Editorial changes to address Adrian’s comments and comments during YANG docotor review.

* Open issue: Are there any risks associated with an attacker adding or removing tags so that a requester gets the wrong data?

v03 - v04
* Remove histogram metric type tag from metric type tags.

* Clarify the object tag and property tag, metric tag are mutual exclusive.

* Clarify to have two optional node tags (i.e., object tag and property tag) to indicate relationship between data nodes.

* Update targeted data node collection example.

v02 - v03

* Additional Editorial changes.

* Security section enhancement.

* Nits fixed.

v01 - v02

* Clarify the relation between data node, object tag, property tag and metric tag in figure 1 and figure 2 and related description;

* Change Metric Group into Metric Type in the YANG model;

* Add 5 metric types in section 7.2;

v00 - v01

* Merge node tag use case section into introduction section as a subsection;

* Add one glossary section;

* Clarify the relation between data node, object tag, property tag and metric tag in node Tags Use Case section;

* Add update to RFC8407 in the front page.

Authors’ Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com
Benoit Claise
Huawei
De Kleetlaan 6a b1
1831 Diegem
Belgium
Email: benoit.claise@huawei.com

Peng Liu
China Mobile
32 Xuanwumen West St, Xicheng District
Beijing
Email: liupengyjy@chinamobile.com

Zongpeng Du
China Mobile
32 Xuanwumen West St, Xicheng District
Beijing
Email: duzongpeng@chinamobile.com

Mohamed Boucadair
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com
A YANG Data Model for Syslog Configuration
draft-ietf-netmod-syslog-model-27

Abstract

This document defines a YANG data model for the configuration of a syslog process. It is intended this model be used by vendors who implement syslog in their systems.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at 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 7 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

This document defines a YANG [RFC7950] configuration data model that may be used to configure the syslog feature running on a system. YANG models can be used with network management protocols such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

The data model makes use of the YANG "feature" construct which allows implementations to support only those syslog features that lie within their capabilities.
This module can be used to configure the syslog application conceptual layers as implemented on the target system.

Essentially, a syslog process receives messages (from the kernel, processes, applications or other syslog processes) and processes them. The processing may involve logging to a local file, and/or displaying on console, and/or relaying to syslog processes on other machines. The processing is determined by the "facility" that originated the message and the "severity" assigned to the message by the facility.

Such definitions of syslog protocol are defined in [RFC5424], and are used in this RFC.

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

1.1. Requirements Language

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

2. Terminology

The term "originator" is defined in [RFC5424]: an "originator" generates syslog content to be carried in a message.

The term "relay" is defined in [RFC5424]: a "relay" forwards messages, accepting messages from originators or other relays and sending them to collectors or other relays.

The term "collectors" is defined in [RFC5424]: a "collector" gathers syslog content for further analysis.

The term "action" refers to the processing that takes place for each syslog message received.

3. NDMA Compliance

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].
4. Editorial Note (To be removed by RFC Editor)

This document contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

* I-D.ietf-netconf-crypto-types --> the assigned RFC value for draft-ietf-netconf-crypto-types

* I-D.ietf-netconf-tls-client-server --> the assigned RFC value for draft-ietf-netconf-tls-client-server

* zzzz --> the assigned RFC value for this draft

5. Design of the Syslog Model

The syslog model was designed by comparing various syslog features implemented by various vendors’ in different implementations.

This document addresses the common leaves between implementations and creates a common model, which can be augmented with proprietary features, if necessary. This model is designed to be very simple for maximum flexibility.

Some optional features are defined in this document to specify functionality that is present in specific vendor configurations.

Syslog consists of originators and collectors. The following diagram shows syslog messages flowing from originators, to collectors where filtering can take place.
Collectors are configured using the leaves in the syslog model "actions" container which correspond to each message collector:

- console
- log file(s)
- remote relay(s)/collector(s)

Within each action, a selector is used to filter syslog messages. A selector consists of a list of one or more filters specified by facility-severity pairs, and, if supported via the select-match feature, an optional regular expression pattern match that is performed on the [RFC5424] field.

A syslog message is processed if:
There is an element of facility-list \((F, S)\) where
   the message facility matches \(F\)
   and the message severity matches \(S\)
   and/or the message text matches the regex pattern (if it is present)

The facility is one of a specific syslog-facility, or all facilities.

The severity is one of type syslog-severity, all severities, or none.
None is a special case that can be used to disable a filter. When filtering severity, the default comparison is that messages of the specified severity and higher are selected to be logged. This is shown in the model as "default equals-or-higher". This behavior can be altered if the select-adv-compare feature is enabled to specify a compare operation and an action. Compare operations are: "equals" to select messages with this single severity, or "equals-or-higher" to select messages of the specified severity and higher. Actions are used to log the message or block the message from being logged.

Many vendors extend the list of facilities available for logging in their implementation. An example is included in Extending Facilities (Appendix A.1).

5.1. Syslog Module

A simplified graphical representation of the data model is used in this document. Please see [RFC8340] for tree diagram notation.
|     |        +--rw compare?   enumeration
|     |        +--rw action?    enumeration
|     |        +--rw pattern-match?  string {select-match}?
|     |        +--rw structured-data? boolean {structured-data}?
|     +--rw file-rotation
|     |        +--rw number-of-files?  uint32 {file-limit-size}?
|     |        +--rw max-file-size?  uint32 {file-limit-size}?
|     +--rw rollover?
|        |       uint32
|     +--rw retention?
|        |       uint32
|     |        +--rw file-limit-duration?
|     +--rw remote {remote-action}?
|     +--rw destination* [name]
|     +--rw name   string
|     +--rw (transport)
|        +--:(udp)
|        |        +--rw udp
|        |           +--rw address?  inet:host
|        |           +--rw port?  inet:port-number
|        +--:(tls)
|        |        +--rw tls
|        |           +--rw address?  inet:host
|        |           +--rw port?
|        |                 inet:port-number
|        |        +--rw client-identity!
|        |        +--rw (auth-type)
|        |        |        +--:(certificate)
|        |        |           +--rw certificate
|        |        |           +--rw (local-or-keystore)
|        |        |           +--:(local)
|        |        |                 {local-definitions-supported, asymmetric-keys}?
|        |        +--rw local-definition
|        |           +--rw public-key-format
|        |             |       identityref
|        |           +--rw public-key
|        |             |       binary
|        |           +--rw private-key-format?
|        |             |       identityref
|        |           +--rw (private-key-type)
|        |           +--:(cleartext-private-key)
|        |           +--: (hidden-private-key)
|        |                  {hidden-keys}?
|     |        +--rw cleartext-private-key?  binary
|        |        +--: (hidden-private-key)
certificates)?
  +--rw truststore-reference?
      ts:certificate-bag-ref
  +--rw ee-certs! {server-auth-x509-cert}?
    +--rw (local-or-truststore)
      +--:(local)
        {local-definitions-supported}?
        +--rw local-definition
          +--rw certificate* [name]
            +--rw name
                string
            +--rw cert-data
                trust-anchor-cert-cms
            +--n certificate-expiration
                (certificate-expiration-notification)?
                +-- expiration-date
                    yang:date-and-time
                +--:(truststore)
                    {central-truststore-supported, certificates}?
                    +--rw truststore-reference?
                        ts:certificate-bag-ref
                    +--rw raw-public-keys!
                        {server-auth-raw-public-key}?
                        +--rw (local-or-truststore)
                          +--:(local)
                            {local-definitions-supported}?
                            +--rw local-definition
                            +--rw public-key* [name]
                               +--rw name
                                   string
                               +--rw public-key-format
                                   identityref
                               +--rw public-key
                                   binary
                            +--:(truststore)
                              {central-truststore-supported, public-keys}?
                              +--rw truststore-reference?
                                  ts:public-key-bag-ref
                              +--rw tls12-psks? empty
                                  {server-auth-tls12-psk}?
                              +--rw tls13-epsk? empty
                                  {server-auth-tls13-epsk}?
                              +--rw hello-params {tlscmn:hello-params}?
                                  +--rw tls-versions
                                     +--rw tls-version* identityref
+-rw cipher-suites
  ++-rw cipher-suite* identityref
+-rw keepalives {tls-client-keepalives}?
  ++-rw peer-allowed-to-send? empty
  ++-rw test-peer-aliveness!
    ++-rw max-wait? uint16
    ++-rw max-attempts? uint8
+-rw facility-filter
  +-rw facility-list* [facility severity]
    ++-rw facility union
    ++-rw severity union
    ++-rw advanced-compare {select-adv-compare}?
      ++-rw compare? enumeration
      ++-rw action? enumeration
  +-rw structured-data? boolean {structured-data}?
  +-rw facility-override? identityref
  +-rw source-interface? if:interface-ref
    {remote-source-interface}?
  +-rw signing! {signed-messages}?
  +-rw cert-signers
    +-rw cert-signer* [name]
      ++-rw name string
      +-rw cert
        +-rw public-key-format
          | identityref
        +-rw public-key
          | binary
        +-rw private-key-format? identityref
        +-rw (private-key-type)
          +--:(cleartext-private-key)
            +-rw cleartext-private-key?
              | binary
          +--:(hidden-private-key) {hidden-keys}?
            +-rw hidden-private-key?
              | empty
          +--:(encrypted-private-key)
            {private-key-encryption}?
              +-rw encrypted-private-key
                +-rw encrypted-by
                  +-rw encrypted-value-format identityref
                  +-rw encrypted-value binary
  +-rw certificates
    +-rw certificate* [name]
      +-rw name
6. Syslog YANG Module

6.1. The ietf-syslog Module

This module imports typedefs from [RFC6991], [RFC8343], groupings from [I-D.ietf-netconf-crypto-types], and [I-D.ietf-netconf-tls-client-server], and it references [RFC5424], [RFC5425], [RFC5426], and [RFC5848], [RFC8089], [RFC8174], and [Std-1003.1-2008].

<CODE BEGINS> file "ietf-syslog@2022-04-05.yang"
module ietf-syslog {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog";
  prefix syslog;

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

  +---ro output
    +---ro certificate-signing-request ct:csr
    +---w input
      |  +---w csr-info ct:csr-info
      +--ro output
        +--ro certificate-signing-request ct:csr
  +--rw hash-algorithm? enumeration
  +---w cert-initial-repeat? uint32
  +---w cert-resend-delay? uint32
  +---w cert-resend-count? uint32
  +--rw sig-max-delay? uint32
  +--rw sig-number-resends? uint32
  +--rw sig-resend-delay? uint32
  +--rw sig-resend-count? uint32

Figure 1: Tree Diagram for Syslog Model
import ietf-interfaces {
  prefix if;
  reference
    "RFC 8343: A YANG Data Model for Interface Management";
}
import ietf-tls-client {
  prefix tlsc;
  reference
    "I-D.iets-netconf-tls-client-server:
        YANG Groupings for TLS Clients and TLS Servers";
}
import ietf-crypto-types {
  prefix ct;
  reference
    "I-D.iets-netconf-crypto-types: YANG Data Types for
        Cryptography";
}

organization
  "IETF NETMOD (Network Modeling) Working Group";
contact
  "WG Web:  <https://datatracker.ietf.org/wg/netmod/>
    WG List:  <mailto:netmod@ietf.org>
    Editor:  Mahesh Jethanandani
        <mailto:mjethanandani@gmail.com>
    Editor:  Joe Clarke
        <mailto:jclarke@cisco.com>
    Editor:  Kiran Agrahara Sreenivasa
        <mailto:kirankoushik.agraharasreenivasa@verizonwireless.com>
    Editor:  Clyde Wildes
        <mailto:cwildes@cisco.com>";
description
  "This module contains a collection of YANG definitions
    for syslog configuration.

Copyright (c) 2022 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 Revised BSD License set
forth in Section 4.c of the IETF Trust’s Legal Provisions
Relating to IETF Documents"
This version of this YANG module is part of RFC zzzz
(https://www.rfc-editor.org/info/rfczzzz); see the RFC itself
for full legal notices.

‘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.

revision 2022-04-05 {
  description
    "Initial Revision";
  reference
    "RFC zzzz: Syslog YANG Model";
}

feature console-action {
  description
    "This feature indicates that the local console action is
    supported.";
}

feature file-action {
  description
    "This feature indicates that the local file action is
    supported.";
}

feature file-limit-size {
  description
    "This feature indicates that file logging resources
    are managed using size and number limits.";
}

feature file-limit-duration {
  description
    "This feature indicates that file logging resources
    are managed using time based limits.";
}

feature remote-action {
  description
    "This feature indicates that the remote server action is
    supported.";
}
feature remote-source-interface {
    description "This feature indicates that source-interface is supported for the remote-action."
}

feature select-adv-compare {
    description "This feature represents the ability to select messages using the additional comparison operators when comparing the syslog message severity."
}

feature select-match {
    description "This feature represents the ability to select messages based on a Posix 1003.2 regular expression pattern match."
}

feature structured-data {
    description "This feature represents the ability to log messages in structured-data format."
    reference "RFC 5424: The Syslog Protocol"
}

feature signed-messages {
    description "This feature represents the ability to configure signed syslog messages."
    reference "RFC 5848: Signed Syslog Messages"
}

typedef syslog-severity {
    type enumeration {
        enum emergency {
            value 0;
            description "The severity level 'Emergency' indicating that the system is unusable."
        }
        enum alert {
            value 1;
            description "The severity level 'Alert' indicating that an action must be taken immediately."
        }
    }
}
enum critical {
    value 2;
    description
        "The severity level 'Critical' indicating a critical condition.";
}

enum error {
    value 3;
    description
        "The severity level 'Error' indicating an error condition.";
}

enum warning {
    value 4;
    description
        "The severity level 'Warning' indicating a warning condition.";
}

enum notice {
    value 5;
    description
        "The severity level 'Notice' indicating a normal but significant condition.";
}

enum info {
    value 6;
    description
        "The severity level 'Info' indicating an informational message.";
}

enum debug {
    value 7;
    description
        "The severity level 'Debug' indicating a debug-level message.";
}

description
    "The definitions for Syslog message severity.
    Note that a lower value is a higher severity. Comparisons of equal-or-higher severity mean equal or lower numeric value";
reference
    "RFC 5424: The Syslog Protocol";

identity syslog-facility {
    description

"This identity is used as a base for all syslog facilities.";
reference
"RFC 5424: The Syslog Protocol";
)

identity kern {
base syslog-facility;
description
"The facility for kernel messages (0).";
reference
"RFC 5424: The Syslog Protocol";
}

identity user {
base syslog-facility;
description
"The facility for user-level messages (1).";
reference
"RFC 5424: The Syslog Protocol";
}

identity mail {
base syslog-facility;
description
"The facility for the mail system (2).";
reference
"RFC 5424: The Syslog Protocol";
}

identity daemon {
base syslog-facility;
description
"The facility for the system daemons (3).";
reference
"RFC 5424: The Syslog Protocol";
}

identity auth {
base syslog-facility;
description
"The facility for security/authorization messages (4).";
reference
"RFC 5424: The Syslog Protocol";
}

identity syslog {
base syslog-facility;
description
"The facility for messages generated internally by syslogd facility (5).";
reference
"RFC 5424: The Syslog Protocol";

identity lpr {
  base syslog-facility;
  description
  "The facility for the line printer subsystem (6).";
  reference
  "RFC 5424: The Syslog Protocol";
}

identity news {
  base syslog-facility;
  description
  "The facility for the network news subsystem (7).";
  reference
  "RFC 5424: The Syslog Protocol";
}

identity uucp {
  base syslog-facility;
  description
  "The facility for the UUCP subsystem (8).";
  reference
  "RFC 5424: The Syslog Protocol";
}

identity cron {
  base syslog-facility;
  description
  "The facility for the clock daemon (9).";
  reference
  "RFC 5424: The Syslog Protocol";
}

identity authpriv {
  base syslog-facility;
  description
  "The facility for privileged security/authorization messages (10).";
  reference
  "RFC 5424: The Syslog Protocol";
}

identity ftp {

base syslog-facility;
  description
    "The facility for the FTP daemon (11).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity ntp {
  base syslog-facility;
  description
    "The facility for the NTP subsystem (12).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity audit {
  base syslog-facility;
  description
    "The facility for log audit messages (13).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity console {
  base syslog-facility;
  description
    "The facility for log alert messages (14).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity cron2 {
  base syslog-facility;
  description
    "The facility for the second clock daemon (15).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity local0 {
  base syslog-facility;
  description
    "The facility for local use 0 messages (16).";
  reference
    "RFC 5424: The Syslog Protocol";
}

identity local1 {
identity local2 {
    base syslog-facility;
    description
        "The facility for local use 2 messages (18).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local3 {
    base syslog-facility;
    description
        "The facility for local use 3 messages (19).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local4 {
    base syslog-facility;
    description
        "The facility for local use 4 messages (20).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local5 {
    base syslog-facility;
    description
        "The facility for local use 5 messages (21).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local6 {
    base syslog-facility;
    description
        "The facility for local use 6 messages (22).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local7 {
base syslog-facility;
description
  "The facility for local use 7 messages (23).";
reference
  "RFC 5424: The Syslog Protocol";
}

grouping severity-filter {
  description
  "This grouping defines the processing used to select
log messages by comparing syslog message severity using
the following processing rules:
- if 'none', do not match.
- if 'all', match.
- else compare message severity with the specified severity
  according to the default compare rule (all messages of the
  specified severity and greater match) or if the
  select-adv-compare feature is present, use the
  advance-compare rule.";
leaf severity {
  type union {
    type syslog-severity;
    type enumeration {
      enum none {
        value 2147483647;
        description
          "This enum describes the case where no severities
          are selected.";
      }
      enum all {
        value -2147483648;
        description
          "This enum describes the case where all severities
          are selected.";
      }
    }
  }
  mandatory true;
description
  "This leaf specifies the syslog message severity.";
}
container advanced-compare {
  when "./severity != "all" and
    ../severity != "none"
  { description
    "The advanced compare container is not applicable for
    severity 'all' or severity 'none'";
  }
}
if-feature "select-adv-compare";
leaf compare {
    type enumeration {
        enum equals {
            description
                "This enum specifies that the severity comparison
                operation will be equals.";
        }
        enum equals-or-higher {
            description
                "This enum specifies that the severity comparison
                operation will be equals or higher.";
        }
    }
    default "equals-or-higher";
    description
        "The compare can be used to specify the comparison
        operator that should be used to compare the syslog message
        severity with the specified severity.";
}
leaf action {
    type enumeration {
        enum log {
            description
                "This enum specifies that if the compare operation is
                true the message will be logged.";
        }
        enum block {
            description
                "This enum specifies that if the compare operation is
                true the message will not be logged.";
        }
    }
    default "log";
    description
        "The action can be used to specify if the message should
        be logged or blocked based on the outcome of the compare
        operation.";
}
grouping selector {
  description
  "This grouping defines a syslog selector which is used to
  select log messages for the log-actions (console, file,
  remote, etc.). Choose one or both of the following:
  facility [<facility> <severity>...]
  pattern-match regular-expression-match-string
  If both facility and pattern-match are specified, both must
  match in order for a log message to be selected.";
  container facility-filter {
    description
    "This container describes the syslog filter parameters.";
    list facility-list {
      key "facility severity";
      ordered-by user;
      description
      "This list describes a collection of syslog
      facilities and severities.";
      leaf facility {
        type union {
          type identityref {
            base syslog-facility;
          }
          type enumeration {
            enum all {
              description
              "This enum describes the case where all
              facilities are requested.";
            }
          }
        }
        description
        "The leaf uniquely identifies a syslog facility.";
      }
    }
    uses severity-filter;
  }
  leaf pattern-match {
    if-feature "select-match";
    type string;
    description
    "This leaf describes a Posix 1003.2 regular expression
    string that can be used to select a syslog message for
    logging. The match is performed on the SYSLOG-MSG field.";
    reference
  }
}
grouping structured-data {
  description "This grouping defines the syslog structured data option which is used to select the format used to write log messages.";
  leaf structured-data {
    if-feature "structured-data";
    type boolean;
    default "false";
    description "This leaf describes how log messages are written. If true, messages will be written with one or more STRUCTURED-DATA elements; if false, messages will be written with STRUCTURED-DATA = NILVALUE.";
    reference "RFC 5424: The Syslog Protocol";
  }
}

container syslog {
  presence "Enables logging.";
  description "This container describes the configuration parameters for syslog.";
  container actions {
    description "This container describes the log-action parameters for syslog.";
    container console {
      if-feature "console-action";
      presence "Enables logging to the console";
      description "This container describes the configuration parameters for console logging.";
      uses selector;
    }
    container file {
      if-feature "file-action";
      description "This container describes the configuration parameters for file logging. If file-archive limits are not supplied, it is assumed that the local implementation defined limits will be used.";
    }
  }
}
list log-file {
  key "name";
  description "This list describes a collection of local logging files.";
  leaf name {
    type inet:uri {
      pattern 'file:.*';
    }
    description "This leaf specifies the name of the log file which MUST use the uri scheme file:";
    reference "RFC 8089: The file URI Scheme";
  }
  uses selector;
  uses structured-data;
  container file-rotation {
    description "This container describes the configuration parameters for log file rotation.";
    leaf number-of-files {
      if-feature "file-limit-size";
      type uint32;
      default "1";
      description "This leaf specifies the maximum number of log files retained. Specify 1 for implementations that only support one log file.";
    }
    leaf max-file-size {
      if-feature "file-limit-size";
      type uint32;
      units "megabytes";
      description "This leaf specifies the maximum log file size.";
    }
    leaf rollover {
      if-feature "file-limit-duration";
      type uint32;
      units "minutes";
      description "This leaf specifies the length of time that log events should be written to a specific log file. Log events that arrive after the rollover period cause the current log file to be closed and a new log file to be opened.";
    }
  }
}
leaf retention {
  if-feature "file-limit-duration";
  type uint32;
  units "minutes";
  description
      "This leaf specifies the length of time that completed/closed log event files should be stored in the file system before they are removed.";
}
}
}

container remote {
  if-feature "remote-action";
  description
      "This container describes the configuration parameters for forwarding syslog messages to remote relays or collectors."
list destination {
  key "name";
  description
      "This list describes a collection of remote logging destinations.";
  leaf name {
    type string;
    description
      "An arbitrary name for the endpoint to connect to.";
  }
choice transport {
  mandatory true;
  description
      "This choice describes the transport option.";
  case udp {
    container udp {
      description
          "This container describes the UDP transport options."
      reference
          "RFC 5426: Transmission of Syslog Messages over UDP";
    leaf address {
      type inet:host;
      description
          "The leaf uniquely specifies the address of the remote host. One of the following must be specified: an ipv4 address, an ipv6 address, or a host name.";
    }
  }
}
leaf port {
  type inet:port-number;
  default "514";
  description
      "This leaf specifies the port number used to
       deliver messages to the remote server."
}
}

case tls {
  container tls {
    description
      "This container describes the TLS transport
       options.";
    reference
      "RFC 5425: Transport Layer Security (TLS)
       Transport Mapping for Syslog ";
    leaf address {
      type inet:host;
      description
        "The leaf uniquely specifies the address of
         the remote host. One of the following must be
         specified: an ipv4 address, an ipv6 address,
         or a host name.";
    }
    leaf port {
      type inet:port-number;
      default "6514";
      description
        "TCP port 6514 has been allocated as the default
         port for syslog over TLS.";
    }
    uses tlsc:tls-client-grouping;
  }
  uses selector;
  uses structured-data;
  leaf facility-override {
    type identityref {
      base syslog-facility;
    }
    description
      "If specified, this leaf specifies the facility used
       to override the facility in messages delivered to
       the remote server.";
  }
  leaf source-interface {
if-feature "remote-source-interface";
type if:interface-ref;
description
   "This leaf sets the source interface to be used to
   send messages to the remote syslog server. If not
   set, messages can be sent on any interface."
}

container signing {
   if-feature "signed-messages";
presence "If present, syslog-signing options is activated.";
description
   "This container describes the configuration
   parameters for signed syslog messages.";
reference
   "RFC 5848: Signed Syslog Messages";
container cert-signers {
   description
      "This container describes the signing certificate
      configuration for Signature Group 0 which covers
      the case for administrators who want all Signature
      Blocks to be sent to a single destination.";
list cert-signer {
   key "name";
description
      "This list describes a collection of syslog
      message signers.";
leaf name {
   type string;
description
      "This leaf specifies the name of the syslog
      message signer.";
}
}
container cert {
   uses ct:asymmetric-key-pair-with-certs-grouping;
description
   "This is the certificate that is periodically
   sent to the remote receiver. The certificate
   is inherently associated with its private
   and public keys.";
}
leaf hash-algorithm {
   type enumeration {
      enum SHA1 {
         value 1;
description
            "This enum describes the SHA1 algorithm.";
      }
      enum SHA256 {

value 2;
description
 "This enum describes the SHA256 algorithm.";
}
}
description
 "This leaf describes the syslog signer hash
 algorithm used.";
}
leaf cert-initial-repeat {
type uint32;
default "3";
description
 "This leaf specifies the number of times each
 Certificate Block should be sent before the first
 message is sent.";
}
leaf cert-resend-delay {
type uint32;
units "seconds";
default "3600";
description
 "This leaf specifies the maximum time delay in
 seconds until resending the Certificate Block.";
}
leaf cert-resend-count {
type uint32;
default "0";
description
 "This leaf specifies the maximum number of other
 syslog messages to send until resending the
 Certificate Block.";
}
leaf sig-max-delay {
type uint32;
units "seconds";
default "60";
description
 "This leaf specifies when to generate a new
 Signature Block. If this many seconds have
 elapsed since the message with the first message
 number of the Signature Block was sent, a new
 Signature Block should be generated.";
}
leaf sig-number-resends {
type uint32;
default "0";
description
"This leaf specifies the number of times a Signature Block is resent. (It is recommended to select a value of greater than 0 in particular when the UDP transport RFC 5426 is used.).";

leaf sig-resend-delay {
  type uint32;
  units "seconds";
  default "5";
  description
  "This leaf specifies when to send the next Signature Block transmission based on time. If this many seconds have elapsed since the previous sending of this Signature Block, resend it.";
}

leaf sig-resend-count {
  type uint32;
  default "0";
  description
  "This leaf specifies when to send the next Signature Block transmission based on a count. If this many other syslog messages have been sent since the previous sending of this Signature Block, resend it. A value of 0 means that you don’t resend based on the number of messages."
}

Figure 2: Sylog YANG Model

7. Usage Examples

7.1. Syslog Configuration for Severity Critical
Enable console logging of syslogs of severity critical

<?xml version="1.0" encoding="UTF-8"?>
<syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
  <actions>
    <console>
      <facility-filter>
        <facility-list>
          <facility>all</facility>
          <severity>critical</severity>
        </facility-list>
      </facility-filter>
    </console>
  </actions>
</syslog>

Figure 3: Syslog Configuration for Severity Critical

7.2. Remote Syslog Configuration
Enable remote logging of syslogs to udp destination foo.example.com for facility auth, severity error

<?xml version="1.0" encoding="UTF-8"?><syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog"><actions><remote><destination><name>remote1</name><udp><address>foo.example.com</address></udp><facility-filter><facility-list><facility>auth</facility><severity>error</severity></facility-list></facility-filter></destination></remote></actions></syslog>

Figure 4: Remote Syslog Configuration

8. Acknowledgements

The authors wish to thank the following who commented on this proposal:

Andy Bierman, Martin Bjorklund, Alex Campbell, Alex Clemm, Francis Dupont, Jim Gibson, Jeffrey Haas, Bob Harold, John Heasley, Giles Heron, Lisa Huang, Mahesh Jethanandani, Warren Kumari, Jeffrey K Lange, Jan Lindblad, Chris Lonvick, Alexey Melnikov, Kathleen Moriarty, Tom Petch, Adam Roach, Juergen Schoenwaelder, Phil Shafer, Yaron Sheffer, Jason Sterne, Peter Van Horne, Kent Watsen, Bert Wijnen, Dale R Worley, and Aleksandr Zhdankin.

9. IANA Considerations

9.1. The IETF XML Registry

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

Clarke, et al. Expires 7 October 2022
9.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC7895]. Following the format in [RFC7950], the following registration is requested:

- name:         ietf-syslog
- prefix:       ietf-syslog
- reference:    RFC zzzz

10. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes should be considered sensitive or vulnerable in all network environments. Logging in particular is used to assess the state of systems and can be used to indicate a network compromise. If logging were to be disabled through malicious means, attacks may not be readily detectable. Therefore write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations and on network security.

In addition there are data nodes that require careful analysis and review. These are the subtrees and data nodes and their sensitivity/vulnerability:

- facility-filter/pattern-match: When writing this node, implementations MUST ensure that the regular expression pattern match is not constructed to cause a regular expression denial of service attack due to a pattern that causes the regular expression implementation to work very slowly (exponentially related to input size).
remote/destination/signing/cert-signer: When writing this subtree, implementations MUST NOT specify a private key that is used for any other purpose.

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

remote/destination/transport: This subtree contains information about other hosts in the network, and the TLS transport certificate properties if TLS is selected as the transport protocol.

remote/destination/signing: This subtree contains information about the syslog message signing properties including signing certificate information.

There are no RPC operations defined in this YANG module.

11. References

11.1. Normative References


11.2. Informative References


11.2. Informative References

Appendix A. Implementer Guidelines

A.1. Extending Facilities

Many vendors extend the list of facilities available for logging in their implementation. Additional facilities may not work with the syslog protocol as defined in [RFC5424] and hence such facilities apply for local syslog-like logging functionality.

The following is an example that shows how additional facilities could be added to the list of available facilities (in this example two facilities are added):
module example-vendor-syslog-types {
    namespace "http://example.com/ns/vendor-syslog-types";
    prefix vendor-syslogtypes;

    import ietf-syslog {
        prefix syslogtypes;
    }

    organization "Example, Inc.";
    contact
        "Example, Inc.  
        Customer Service
        E-mail: syslog-yang@example.com";

description
    "This module contains a collection of vendor-specific YANG type
    definitions for SYSLOG.";

    revision 2017-08-11 {
        description
            "Version 1.0";
        reference
            "Vendor SYSLOG Types: SYSLOG YANG Model";
    }

    identity vendor_specific_type_1 {
        base syslogtypes:syslog-facility;
        description
            "Adding vendor specific type 1 to syslog-facility";
    }

    identity vendor_specific_type_2 {
        base syslogtypes:syslog-facility;
        description
            "Adding vendor specific type 2 to syslog-facility";
    }
}

A.2. Syslog Terminal Output

Terminal output with requirements more complex than the console subtree currently provides, are expected to be supported via vendor extensions rather than handled via the file subtree.
A.3. Syslog File Naming Convention

The syslog/file/log-file/file-rotation container contains configuration parameters for syslog file rotation. This section describes how these fields might be used by an implementer to name syslog files in a rotation process. This information is offered as an informative guide only.

When an active syslog file with a name specified by log-file/name, reaches log-file/max-file-size and/or syslog events arrive after the period specified by log-file/rollover, the logging system can close the file, can compress it, and can name the archive file <log-file/name>.0.gz. The logging system can then open a new active syslog file <log-file/name>.

When the new syslog file reaches either of the size limits referenced above, <log-file/name>.0.gz can be renamed <log-file/name>.1.gz and the new syslog file can be closed, compressed and renamed <log-file/name>.0.gz. Each time that a new syslog file is closed, each of the prior syslog archive files named <log-file/name>.<n>.gz can be renamed to <log-file/name>.<n + 1>.gz.

Removal of archive log files could occur when either or both:

- log-file/number-of-files specified - the logging system can create up to log-file/number-of-files syslog archive files after which, the contents of the oldest archived file could be overwritten.

- log-file/retention specified - the logging system can remove those syslog archive files whose file expiration time (file creation time plus the specified log-file/retention time) is prior to the current time.

Authors’ Addresses

Joe Clarke (editor)
Cisco
United States of America
Email: jclarke@cisco.com

Mahesh Jethanandani (editor)
Kloud Services
United States of America
Email: mjethanandai@gmail.com
Updated YANG Module Revision Handling
draft-ietf-netmod-yang-module-versioning-06

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 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 6020, 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 11, 2023.
1. Introduction

This document defines the foundational pieces of a solution to the YANG module lifecycle problems described in [I-D.ietf-netmod-yang-versioning-reqs]. Complementary documents provide other parts of the solution, 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 and [RFC6020] section 10. 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 and [RFC6020] 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 and [RFC8525]. 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 following terminology:

- 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.
3. Refinements to YANG revision handling

[RFC7950] and [RFC6020] assume, but do 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] and [RFC6020] 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] and [RFC6020], 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 [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 and [RFC6020] section 10 require 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 and [RFC6020] section 10 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] and [RFC6020], 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 and [RFC6020] section 10, updated by the following rules:

- 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.

- 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 since their identifiers may still be referenced by other modules and to ensure that their identifiers are not reused with a different meaning.

- 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.

- 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.

- Any changes (including whitespace or formatting changes) that do not change the semantic meaning of the module are 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 desirable 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.

An author MAY remove a contiguous sequence of entries from the end (i.e., oldest entries) of the revision history. This is acceptable
even if the first remaining (oldest) revision entry in the revision history contains a rev:non-backwards-compatible substatement.

An author MAY remove a contiguous sequence of entries in the revision history as long as the presence or absence of any existing rev:non-backwards-compatible substatements on all remaining entries still accurately reflect the compatibility relationship to their preceding entries remaining in the revision history.

The author MUST NOT remove the first (i.e., newest) revision entry in the revision history.

Example revision history:

```
revision 2020-11-11 {
  rev:revision-label 4.0.0;
  rev:non-backwards-compatible;
}
revision 2020-08-09 {
  rev:revision-label 3.0.0;
  rev:non-backwards-compatible;
}
revision 2020-06-07 {
  rev:revision-label 2.1.0;
}
revision 2020-02-10 {
  rev:revision-label 2.0.0;
  rev:non-backwards-compatible;
}
revision 2019-10-21 {
  rev:revision-label 1.1.3;
}
revision 2019-03-04 {
  rev:revision-label 1.1.2;
}
revision 2019-01-02 {
  rev:revision-label 1.1.1;
}
```

In the revision history example above, removing the revision history entry for 2020-02-10 would also remove the rev:non-backwards-
compatible annotation and hence the resulting revision history would
incorrectly indicate that revision 2020-06-07 is backwards-compatible
with revisions 2019-01-02 through 2019-10-21 when it is not, and so
this change cannot be made. Conversely, removing one or more
revisions out of 2019-03-04, 2019-10-21 and 2020-08-09 from the
revision history would still retain a consistent revision history,
and is acceptable, subject to an awareness of the concerns raised in
the first paragraph of this section.

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.

A revision label scheme is a set of rules describing how a particular
type of revision-label operates for versioning YANG modules and
submodules. For example, YANG Semver [I-D.ietf-netmod-yang-semver]
defines a revision label scheme based on Semver 2.0.0 [semver].
Other documents may define other YANG revision label schemes.

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 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 and [RFC6020] 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:

- specifies the revision-label scheme used by the module or submodule
- is defined in the document which specifies the revision-label scheme
- 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.

<table>
<thead>
<tr>
<th>Module revision date</th>
<th>Revision label</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-01-01</td>
<td>&lt;- 1.0.0</td>
</tr>
<tr>
<td>2019-02-01</td>
<td>&lt;- 2.0.0</td>
</tr>
<tr>
<td>2019-03-01</td>
<td>&lt;- 3.0.0</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>&lt;- 2.1.0</td>
</tr>
<tr>
<td>2019-05-01</td>
<td>&lt;- 2.2.0</td>
</tr>
<tr>
<td>2019-06-01</td>
<td>&lt;- 3.1.0</td>
</tr>
</tbody>
</table>

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

[RFC7950] and [RFC6020] allow 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.

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:

<table>
<thead>
<tr>
<th>Module revision date</th>
<th>Revision label</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-01-01</td>
<td>&lt;- 1.0.0</td>
</tr>
<tr>
<td>2019-02-01</td>
<td>&lt;- 2.0.0</td>
</tr>
<tr>
<td>2019-03-01</td>
<td>&lt;- 3.0.0</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>&lt;- 2.1.0</td>
</tr>
<tr>
<td>2019-05-01</td>
<td>&lt;- 2.2.0</td>
</tr>
<tr>
<td>2019-06-01</td>
<td>&lt;- 3.1.0</td>
</tr>
</tbody>
</table>

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.

```yaml
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.

```yaml
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.

```yaml
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.

```yang
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 1.1 [RFC7950] and YANG library [RFC8525] to clarify how ambiguous module imports are resolved. It also defines the YANG module, ietf-yang-library-revisions, that augments YANG library [RFC8525] with revision labels and two leafs to indicate how a server implements deprecated and obsolete schema nodes.

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] section 5.6.5 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" and "submodule" lists in ietf-yang-library with "revision-label" leafs to optionally declare the revision label associated with each module and submodule.

5.2.2. Reporting how deprecated and obsolete nodes are handled

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

deprecated-nodes-implemented: If set to "true", this leaf indicates that all schema nodes with a status "deprecated" are implemented equivalently as if they had status "current"; 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" schema 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" leafs 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 SHOULD 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.

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:

- 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:

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

- NBC changes are done at once, i.e. without using "status" statements. Depending on the change, this may have a big impact on clients.

- 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". At some point in the future, when support is removed for the data node, there are two options. The first, and preferred, option is to keep the data node definition in the model and change the status to "obsolete". The second option is to simply remove the data node from the model, but this has the risk of breaking modules which import the modified module, and the removed identifier may be accidently reused in a future revision.

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:

- 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.

- 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.

- 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.

```yang
<CODE BEGINS> file "ietf-yang-revisions@2021-11-04.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>
```
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 2021-11-04 {
  rev:revision-label 1.0.0-draft-ietf-netmod-yang-module-versioning-05;
  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 non-backwards-compatible {
    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 backwards compatible 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, non-backwards-compatible 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. One such 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 in this module.

    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-base;
    description
        "The revision label scheme specifies which revision-label scheme
the module or submodule uses.

The mandatory revision-label-scheme-base 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";

}

YANG module with augmentations to YANG Library to revision labels

<CODE BEGINS> file "ietf-yang-library-revisions@2021-11-04.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

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 2021-11-04 {
  rev:revision-label 1.0.0-draft-ietf-netmod-yang-module-versioning-05;
  description
    "Initial revision";
  reference
    "XXXX: Updated YANG Module Revision Handling";
}

// library 1.0 modules-state is not augmented with revision-label

augment "/yanglib:yang-library.yanglib:module-set.yanglib:module" {
  description
    "Add a revision label to module information";
  leaf revision-label {
    type rev:revision-label;
    description
      "The revision label associated with this module revision. The label MUST match the rev:revision-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
    "Add a revision label to submodule information";
  leaf revision-label {
    type rev:revision-label;
    description
      "The revision label associated with this submodule revision. The label MUST match the rev:revision-label value in the specific revision of the submodule included by the module loaded in this module-set.";
  }
}
augment "/yanglib:yang-library.yanglib:module-set/"
  + "yanglib:import-only-module.yanglib:submodule" {
    description
    "Add a revision label to submodule information";
    leaf revision-label {
      type rev:revision-label;
      description
      "The revision label associated with this submodule revision. 
The label MUST match the rev:revision-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:module-set/"
  + "yanglib:import-only-module.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' are implemented
equivalently as if they had status 'current'; 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' schema 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";
}

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:

- Balazs Lengyel
- Benoit Claise
- Bo Wu
- Ebben Aries
- Jan Lindblad
The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update]. We would like to thank Kevin D’Souza and Benoit Claise for their initial work in this problem space.

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] and [RFC6020].

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.

Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
The following YANG module is requested to be registered 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
Prefix: rev
Reference: [RFCXXXX]

The ietf-yang-library-revisions module:

Name: 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 or 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 or 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-13 (work in progress), March 2022.
[I-D.ietf-netmod-yang-semver]


12.2. Informative References


[I-D.ietf-netmod-yang-instance-file-format]

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

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

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

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

[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>.


[RoutingTypesDecRevision]
"2020-12-31 revision of iana-routing-types.yang", <https://www.iana.org/assignments/iana-routing-types@2020-12-31.yang>.

[RoutingTypesYang]

Appendix A. Examples of changes that are NBC

Examples of NBC changes include:

- Deleting a data node, or changing it to status obsolete.
- Changing the name, type, or units of a data node.
- Modifying the description in a way that changes the semantic meaning of the data node.
- 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.
- Adding or modifying "when" statements that reduce when the data node is available in the schema.
- Making the statement conditional on if-feature.

Appendix B. Examples of applying the NBC change guidelines

The following sections give steps that could be taken for making NBC changes to a YANG module or submodule using the incremental approach described in section Section 7.1.1.

The examples are all for "config true" nodes.

Alternatively, the NBC changes MAY be done non-incrementally and without using "status" statements if the server can support multiple revisions of the YANG module or of YANG packages. Clients would be required to select the revision which they support and the NBC change would have no impact on them.

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 schema node’s status is changed to "deprecated" and the node is supported for some period of time (e.g. one year). This is a BC change.

2. When the schema node is not supported anymore, its status is changed to "obsolete" and the "description" updated. This is an NBC change.

B.2. Changing the type of a leaf node

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

1. The status of schema node "vpn-id" is changed to "deprecated" and the node is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate that "vpn-name" is replacing this node.

2. A new schema 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 schema nodes are supported, the interactions between the two nodes 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). A "choice" construction could be used, or 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. The behavior in this case is outside the scope of this document.

4. When the schema node "vpn-id" is not supported anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.
B.3. Reducing the range of a leaf node

Reducing the range of values of a leaf-node, e.g., consider a "vpn-id" schema node of type uint32 being changed from range 1..5000 to range 1..2000:

1. If all values which are being removed were never supported, e.g., if a vpn-id of 2001 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 3333 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 is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate the new list that is replacing this list.

2. A new list is created in the same location with the same descendant schema nodes but with "dest-address" as key. Finding an appropriate name for the new list can be difficult. 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 supported, the interactions between the two lists 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 entries in the new list from being created if the old list already has entries (and vice-versa).

   2. If the new list has entries created 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 cannot 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 or container schema node 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 is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate the node that is replacing this node.

2. The new schema 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, the interactions between the two nodes 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). A "choice" construction could be used, or 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 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-address".

4. When node "ip-address" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

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
Abstract

This document specifies a scheme and guidelines for applying an extended set of semantic versioning rules to revisions of YANG artifacts (e.g., modules and packages). Additionally, this document defines an RFCAAAAA-compliant revision-label-scheme for this YANG semantic versioning 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 11 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

[I-D.ietf-netmod-yang-module-versioning] puts forth a number of concepts relating to modified rules for updating YANG 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 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 3.4 of
This document defines a revision-label scheme that uses extended semantic versioning rules [SemVer] 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 being to add a human readable revision label that provides compatibility information for the YANG artifact without 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] are examples of YANG artifacts for the purposes of this document.

* YANG Semver: A revision-label identifier that is consistent with the extended set of semantic versioning rules, based on [SemVer], defined within this document.

3. YANG Semantic Versioning

This section defines YANG Semantic Versioning, explains how it is used with YANG artifacts, and describes the rules associated with changing an artifact’s semantic version when its contents are updated.
3.1. YANG Semver 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 or equal to 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 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 when comparing YANG semantic versions, pre-release metadata MUST be used during module and submodule development as specified in Section 5. Both pre-release and build metadata are allowed in order to support all 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.

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 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 YANG Semver will reflect that in the package metadata.

As stated above, the YANG semantic version is expressed as a string of the form: ’X.Y.Z_COMPAT’.

* ’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 ‘COMPAT’ modifier string is "sticky". Once a revision of a module has a modifier in the revision label, then all descendants of that revision with the same X.Y version digits will also have a modifier. The modifier can change from "_compatible" to "_non_compatible" in a descendant revision, but the modifier MUST NOT change from "_non_compatible" to "_compatible" and MUST NOT be removed. The persistence of the "_non_compatible" modifier ensures that comparisons of revision labels do not give the false impression of compatibility between two potentially non-compatible revisions. If "_non_compatible" was removed, for example between revisions "3.3.2_non_compatible" and "3.3.3" (where "3.3.3" was simply an editorial change), then comparing revision labels of "3.3.3" back to an ancestor "3.0.0" would look like they are backwards compatible when they are not (since "3.3.2_non_compatible" was in the chain of ancestors and introduced a non-backwards-compatible change).
The YANG artifact name and YANG semantic version 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 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. YANG Semver with submodules

YANG Semver MAY be used to version submodules. Submodule version are separate of any version on the including module, but if a submodule has changed, then the version of the including module MUST also be updated.

The rules for determining the version change of a submodule are the same as those defined in Section 3.1 and Section 3.2 as applied to YANG modules, except they only apply to the part of the module schema defined within the submodule’s file.

One interesting case is moving definitions from one submodule to another in a way that does not change the resultant schema of the including module. In this case:

1. The including module has editorial changes
2. The submodule with the schema definition removed has non-backwards-compatible changes
3. The submodule with the schema definitions added has backwards-compatible changes

Note that the meaning of a submodule may change drastically despite having no changes in content or revision due to changes in other submodules belonging to the same module (e.g. groupings and typedefs declared in one submodule and used in another).

3.2.2. Examples for YANG semantic versions

The following diagram and explanation illustrate how YANG semantic versions work.

YANG Semantic versions for an example module:
The tree diagram above illustrates how the version history might evolve for an example module. The tree diagram only shows the parent/child ancestry relationships between the revisions. It does not describe the chronology of the revisions (i.e. when in time each revision was published relative to the other revisions).

The following description lists an example of what the chronological order of the revisions could look like, from oldest revision to newest:

0.1.0 - first pre-release module version
0.2.0 - second pre-release 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)
2.0.0 - change existing model for performance reasons, e.g. re-key list (NBC)
1.3.0 - improve existing functionality, added leaf "foo-64" (BC)
1.1.1_compatible - backport "foo-64" leaf to 1.1.x to avoid implementing "baz" from 1.2.0. This revision was created after 1.2.0 otherwise it may have been released as 1.2.0. (BC)
3.0.0 - NBC bugfix, rename "baz" to "bar"; also add new BC leaf "wibble"; (NBC)
1.3.1_non_compatible - backport NBC fix, rename "baz" to "bar"  
(NBC)

1.2.1_non_compatible - backport NBC fix, rename "baz" to "bar"  
(NBC)

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

1.4.0 - introduce new leaf "ghoti" (BC)

3.1.0 - introduce new leaf "wobble" (BC)

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

The partial ancestry relationships based on the semantic versioning  
numbers are as follows:

1.0.0 < 1.1.0 < 1.2.0 < 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

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 1.3.1_non_compatible

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 1.4.0

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 does not indicate ancestry. The  
module definition in "2.0.0", for example, does not contain all the  
contents of "1.3.0". Version "2.0.0" is not derived from "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 for the new artifact  
revision is calculated, based on the changes between the two artifact  
revisions, and the YANG semantic version 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:
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 pre-release 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. Additional pre-release rules for modules that have had at least one release are specified in Section 5.
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:non-backwards-compatible" 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:non-backwards-compatible" statement then that MUST be reflected in any YANG semantic 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:non-backwards-compatible" 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 "ysver:yang-semver";

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

  description
      "to be completed";

  revision 2017-08-30 {
    description "Backport 'wibble' leaf";
    rev:revision-label 1.2.2_non_compatible;
  }

  revision 2017-07-30 {
    description "Rename 'baz' to 'bar'";
    rev:revision-label 1.2.1_non_compatible;
    rev:non-backwards-compatible;
  }

  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 2017-02-07 {
    description "First release version.";
    rev:revision-label 1.0.0;
  }

  // Note: semver rules do not apply to 0.X.Y labels.
  // The following pre-release revision statements would not
  // appear in any final published version of a module. They
  // are removed when the final version is published.
  // During the pre-release phase of development, only a
  // single one of these revision statements would appear

  // revision 2017-01-30 {
  //   description "NBC changes to initial revision";
  //   rev:revision-label 0.2.0;
  
  }
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.

```yang
{
  "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. The YANG Semver revision-label value can be used as the argument to `rev:revision-or-derived`. When used as such, any module that contains exactly the same YANG semantic version in its revision history may be used to satisfy the import requirement. For example:

```yang
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 ‘.’ or ‘-’ 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 semantic 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 semantic version 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.

Development of a new module or submodule 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.

For IETF YANG modules and submodules that have already been published, revision-labels MUST be retroactively 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. 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 and the identity to signal its use.
module ietf-yang-semver {
  yang-version 1.1;
  prefix ysver;
  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>
    "Author:   Robert Wilton <mailto:rwilton@cisco.com>
    "Author:   Reshad Rahman <mailto:reshad@yahoo.com>
    "Author:   Balazs Lengyel <mailto:balazs.lengyel@ericsson.com>
    "Author:   Jason Sterne <mailto:jason.sterne@nokia.com>
    "Author:   Benoit Claise <mailto:benoit.claise@huawei.com>"

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

  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.";

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

revision 2021-11-04 {
  rev:revision-label "1.0.0-draft-ietf-netmod-yang-semver-05";
  description "Initial revision";
  reference "RFC XXXX: YANG Semantic Versioning.";
}

identity yang-semver {
  base rev:revision-label-scheme-base;
  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.";
}

typedef version {
  type rev:revision-label {
  }
  description "Represents a YANG semantic version. The rules governing the use of this revision label scheme are defined in the reference for this typedef.";
  reference "RFC XXXX: YANG Semantic Versioning.";
}
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, Bo Wu, Ebben Aries, Jan Lindblad, Jason Sterne, Joe Clarke, Juergen Schoenwaelder, Mahesh Jethanandani, Michael (Wangzitao), Qin Wu, Reshad Rahman, and Rob Wilton.

The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update] . We would like thank Kevin D’Souza for his initial work in this problem space.

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

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


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-semver module:

Name: ietf-yang-semver


Prefix: ysver
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 retroactively 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 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 changes are made.

Given that IANA maintained YANG modules 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

10.2. Informative References

[I-D.clacla-netmod-yang-model-update]

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

[I-D.ietf-netmod-yang-schema-comparison]
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 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-exmod-enhancements-00
1.1.0-draft-jdoe-netmod-exmod-enhancements-01 || 1.1.0-draft-jadoe-netmod-enhancements-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

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
YANG Extension and Metadata Annotation for Immutable Flag
draft-ma-netmod-immutable-flag-02

Abstract

This document defines a YANG extension named "immutable" to indicate that specific "config true" data nodes are not allowed to be created/deleted/updated. To indicate that specific instances of a list/leaf-list node cannot be changed after initialization, a metadata annotation with the same name is also defined. Any data node or instance marked as immutable is read-only to the clients of YANG-driven management protocols, such as NETCONF, RESTCONF and other management operations (e.g., SNMP and CLI requests).

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 5 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

YANG [RFC7950] is a data modeling language used to model both state and configuration data, based on the "config" statement. However, there exists data that should not be modifiable by the client, but still needs to be declared as "config true" to:

* allow configuration of data nodes under immutable lists or containers;

* ensure the existence of specific list entries that are provided and needed by the system, while additional list entries can be created, modified or deleted;
* place "when", "must" and "leafref" constraints between
  configuration and immutable schema nodes.

E.g., the interface name and type values created by the system due to
the hardware currently present in the device cannot be modified by
clients, while configurations such as MTU created by the system are
free to be modified by the client. Further examples and use-cases
are described in Appendix A.

Allowing some configuration to be modifiable while other parts are
not is inconsistent and introduces ambiguity to clients.

To address this issue, this document defines a YANG extension and a
metadata annotation [RFC7952] named "immutable" to indicate the
immutability characteristic of a particular schema node or
instantiated data node. If a schema node is marked as immutable,
data nodes based on the schema MUST NOT be added, removed or updated
by management protocols, such as NETCONF, RESTCONF or other
management operations (e.g., SNMP and CLI requests). If an
instantiated data node is marked as immutable the server MUST reject
changes to it by YANG-driven management protocols, such as NETCONF,
RESTCONF and other management operations (e.g., SNMP and CLI
requests). Marking instance data nodes as immutable (as opposed to
marking schema-nodes) is important when only some instances of a list
or leaf-list shall be marked as read-only.

Theoretically, any "config true" data node is allowed to be created,
updated and deleted. This work makes write access restrictions other
than general YANG and NACM rules visible, which doesn’t mean
attaching such restrictions is encouraged.

1.1. Terminology

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

The following terms are defined in [RFC6241] and [RFC8341] and are
not redefined here:

* configuration data

* access operation

* write access
The following terms are defined in this document:

immutable: A property indicating that a schema node or data instance is not allowed to be created/deleted/updated.

2. Overview

The "immutable" concept only puts write access restrictions to read-write datastores. When a specific data node or instance is marked as "immutable", NACM cannot override this to allow create/delete/update access.

A particular data node or instance MUST have the same immutability in all read-write datastores. The immutable annotation information should be visible even in read-only datastores (e.g., <system>, <intended>, <operational>), however this only serves as information about the data node itself, but has no effect on the handling of the read-only datastore. The immutability property of a particular data node or instance MUST be protocol-independent and user-independent.

If a particular container/list/leaf-list node is marked as "immutable" without exceptions for "delete" in the schema, the server SHOULD NOT annotate its instances, as that provides no additional information. If a particular leaf/anydata/anyxml node is marked as "immutable" without exceptions for "delete" or "update" in the schema, the server SHOULD NOT annotate its instances, as that provides no additional information.

Servers MUST reject any attempt to the "create", "delete" and "update" access operations on an immutable data node or instance marked by the metadata annotation or YANG extension (except according to the exceptions argument). The error reporting is performed immediately at an <edit-config> operation time, regardless what the target configuration datastore is. For an example of an "invalid-value" error response, see Appendix A.1.2.

However the following operations SHOULD be allowed:

* Use a create, update, delete/remove operation on an immutable node/instance if the effective change is null. E.g. If a leaf has a current value of "5" it should be allowed to replace it with a value of "5".

* Create an immutable data node/instance with a same value initially set by the system if it doesn’t exist in the datastore. E.g., explicitly configure a system-generated interface name and type in <running>;
Note that even if a particular data node is immutable without the exception for "delete", it still can be deleted with its parent node, e.g., /if:interfaces/if:interface/if:type leaf is immutable, but the deletion to the /if:interfaces/if:interface list entry is allowed; if a particular data node is immutable without the exception for "create", it means the client can never create the instance of it, regardless the handling of its parent node.

TODO: Is immutable inherited down the containment hierarchy? If it is, should we allow overriding the immutability of a particular contained element (i.e., to declare a contained data node as immutable=false inside an immutable container/list)?

3. "Immutable" YANG Extension

The "immutable" YANG extension can be a substatement to a leaf, leaf-list, container, list, anydata or anyxml statement. It indicates that data nodes based on the parent statement MUST NOT be added, removed or updated except according to the exceptions argument. The server MUST reject any such write attempt.

The "immutable" YANG extension defines an argument statement named "exceptions" which gives a list of operations that users are permitted to invoke for the specified node.

The following values are supported for the "exceptions" argument:

* Create: allow users to create instances of the data node;
* Update: allow users to modify instances of the data node;
* Delete: allow users to delete instances of the data node.

4. "Immutable" Metadata Annotation

The "immutable" flag is used to indicate the immutability of a particular instantiated data node. It only applies to the list/leaf-list entries. The values are boolean types indicating whether the data node instance is immutable or not.

Any list/leaf-list instance annotated with immutable="true" is read-only to clients, which means that once an instance is created, the client cannot change it. If a list entry is annotated with immutable="true", any contained descendant instances of any type (including leaves, lists, containers, etc.) inside the specific instance is not allowed to be created, updated and deleted without the need to annotate descendant nodes instances explicitly.
Note that "immutable" metadata annotation is used to annotate instances of a list/leaf-list rather than schema nodes. For instance, a list node may exist in multiple instances in the data tree, "immutable" can annotate some of the instances as read-only, while others are not.

When the client retrieves a particular datastore, immutable data node instances MUST be annotated with immutable="true" by the server. If the "immutable" metadata annotation inside a list entry is not specified, the default "immutable" value for a list/leaf-list entry is false.

Different from the "immutable" YANG extension, deletion to an instance marked with immutable="true" metadata annotation SHOULD always be allowed unless the list/leaf-list data node in the schema has an im:immutable extension as substatement without a "delete" exception.

5. YANG Module

```yaml
<CODE BEGINS>
file="ietf-immutable@2022-04-18.yang"
// RFC Ed.: replace XXXX with RFC number and remove this note
module ietf-immutable {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-immutable";
  prefix im;

  import ietf-yang-metadata {
    prefix md;
  }

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

  contact
    "WG Web: <https://datatracker.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>
  
    Author: Qiufang Ma
    <mailto:maqiufang1@huawei.com>
  
    Author: Qin Wu
    <mailto:bill.wu@huawei.com>
  
    Author: Balazs Lengyel
    <mailto:balazs.lengyel@ericsson.com>
```
This module defines a metadata annotation named 'immutable' to indicate the immutability of a particular instantiated data node. Any instantiated data node marked with immutable='true' by the server is read-only to the clients of YANG-driven management protocols, such as NETCONF, RESTCONF as well as SNMP and CLI requests.

The module defines the immutable extension that indicates that data nodes based on a data-definition statement cannot be added, removed, or updated except according to the exceptions argument.

Copyright (c) 2022 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 Revised 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 HHHH (https://www.rfc-editor.org/info/rfcHHHH); 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.

revision 2022-04-18 {
    description
        "Initial revision.";
    reference
        "RFC XXXX: Immutable Metadata Annotation";
}

extension immutable {
    argument exceptions;
    description
        "The 'immutable' extension as a substatement to a data
definition statement indicates that data nodes based on the parent statement MUST NOT be added, removed or updated by management protocols, such as NETCONF, RESTCONF or other management operations (e.g., SNMP and CLI requests) except when indicated by the exceptions argument.

Immutable data MAY be marked as config true to allow 'leafref', 'when' or 'must' constraints to be based on it.

The statement MUST only be a substatement of the leaf, leaf-list, container, list, anydata, anyxml statements. Zero or one immutable statement per parent statement is allowed. No substatements are allowed.

The argument is a list of operations that are permitted to be used for the specified node, while other operations are forbidden by the immutable extension.
- create: allows users to create instances of the data node
- update: allows users to modify instances of the data node
- delete: allows users to delete instances of the data node

To disallow all user write access, omit the argument;

To allow only create and delete user access, provide the string 'create delete' for the 'exceptions' parameter.

Providing all 3 parameters has the same affect as not using this extension at all, but can be used anyway.

Equivalent YANG definition for this extension:

leaf immutable {
  type bits {
    bit create;
    bit update;
    bit delete;
  }
  default ''; 
}

Adding immutable or removing values from the exceptions argument of an existing immutable statement are non-backwards compatible changes. Other changes to immutable are backwards compatible.";
md:annotation immutable {
  type boolean;
  description
  "The 'immutable' annotation indicates the immutability of an
  instantiated data node. Any data node instance marked as
  'immutable=true' is read-only to clients and cannot be
  updated through NETCONF, RESTCONF or CLI. It applies to the
  list and leaf-list entries. The default is 'immutable=false'
  if not specified for an instance."
}

6. IANA Considerations

6.1. The "IETF XML" Registry

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

  Registrant Contact: The IESG.
  XML: N/A, the requested URIs are XML namespaces.

6.2. The "YANG Module Names" Registry

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

name: ietf-immutable
  prefix: im
  RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

7. Security Considerations

The YANG module specified in this document defines a metadata
annotation for data nodes that is designed to be accessed network
management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].
The lowest NETCONF layer is the secure transport layer, and the
mandatory-to-implement secure transport is Secure Shell (SSH)
[RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-
implement secure transport is TLS [RFC8446].
Since immutable information is tied to applied configuration values, it is only accessible to clients that have the permissions to read the applied configuration values.

The security considerations for the Defining and Using Metadata with YANG (see Section 9 of [RFC7952]) apply to the metadata annotation defined in this document.

8. References

8.1. Normative References


8.2. Informative References

[I-D.ma-netmod-with-system]


Appendix A. Usage Examples

A.1. Interface Example

This section shows how to use im:immutable YANG extension to mark some data node as immutable.

When an interface is physically present, the system will create an interface entry automatically with valid name and type values in <system> (see [I-D.ma-netmod-with-system]). The system-generated data is dependent on and must represent the HW present, and as a consequence must not be changed by the client. The data is modelled as "config true" and should be marked as immutable.

Seemingly an alternative would be to model the list and these leaves as "config false", but that does not work because:

* The list cannot be marked as "config false", because it needs to contain configurable child nodes, e.g., ip-address or enabled;

* The key leaf (name) cannot be marked as "config false" as the list itself is config true;

* The type cannot be marked "config false", because we MAY need to reference the type to make different configuration nodes conditionally available.
The immutability of the data is the same for all interface instances, thus following fragment of a fictional interface module including an "immutable" YANG extension can be used:

```yang
container interfaces {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf type {
      im:immutable "create";
      type identityref {
        base ianaift:iana-interface-type;
      }
      mandatory true;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
    }
  }
}
```

Note that the "name" leaf is defined as a list key which can never been modified for a particular list entry, there is no need to mark "name" as immutable.

A.1.1. Creating an Interface with a "type" Value

As defined in the YANG model, there is an exception for "create" operation. Assume the interface hardware is not present physically at this point, the client is allowed to create an interface named "eth0" with a type value in <running>: 

```bash
```
The interface data does not appear in <operational> since the physical interface doesn’t exist. When the interface is inserted, the system will detect it and create the associated configuration in <system>. The system tries to merge the interface configuration in the <running> datastore with the same name as the inserted interface configuration in <system>. If no such interface configuration named "eth0" is found in <system> or the type set by the client doesn’t match the real interface type generated by the system, only the system-defined interface configuration is applied and present in <operational>.

A.1.2. Updating the Value of an Interface Type

Assume the system applied the interface configuration named "eth0" successfully. If a client tries to change the type of an interface to a value that doesn’t match the real type of the interface used by the system, the server must reject the request:
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interface xc:operation="merge"
                xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
        <name>eth0</name>
        <type>ianaift:tunnel</type>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
            xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path xmlns:t="http://example.com/schema/1.2/config">
      /interfaces/interface[name="eth0"]/type
    </error-path>
    <error-message xml:lang="en">
      Invalid type for interface eth0
    </error-message>
  </rpc-error>
</rpc-reply>

A.2. Immutable System Capabilities Modelled as "config true"

System capabilities might be represented as system-defined data nodes in the model. Configurable data nodes might need constraints specified as "when", "must" or "path" statements to ensure that configuration is set according to the system’s capabilities. E.g.,

* A timer can support the values 1,5,8 seconds. This is defined in the leaf-list 'supported-timer-values'.

* When the configurable 'interface-timer' leaf is set, it should be ensured that one of the supported values is used. The natural solution would be to make the 'interface-timer' a leaf-ref pointing at the 'supported-timer-values'.
However, this is not possible as 'supported-timer-values' must be read-only thus config=false while 'interface-timer' must be writable thus config=true. According to the rules of YANG it is not allowed to put a constraint between config true and false schema nodes.

The solution is that the supported-timer-values data node in the YANG Model shall be defined as "config true" and shall also be marked with the "immutable" extension. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

A.3. Immutable System-defined List Entries

There are some system-defined entries for a "config true" list which are present in <system> (see [I-D.ma-netmod-with-system]) and cannot be updated by the client, such system-defined instances should be defined immutable. The client is free to define, update and delete their own list entries in <running>. Thus the list data node in the YANG model cannot be marked as "immutable" extension as a whole. But some of the system-defined list entries need to be protected if they are copied from the <system> datastore to <running>.

An immutable metadata annotation can be useful in this case. When the client retrieves those system-defined entries towards <system> (or <running> if they are copied into <running>), an immutable="true" annotation is returned; so that the client can understand that the predefined list entries shall not be updated but they can configure their list entries without any restriction.

Appendix B. Changes between revisions

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

v01 - v02

* clarify the relation between the creation/deletion of the immutable data node with its parent data node;

* Add a "TODO" comment about the inheritance of the immutable property;

* Define that the server should reject write attempt to the immutable data node at an <edit-config> operation time, rather than waiting until a <commit> or <validate> operation takes place;

v00 - v01

* Added immutable extension
* Added new use-cases for immutable extension and annotation
* Added requirement that an update that means no effective change should always be allowed
* Added clarification that immutable is only applied to read-write datastore
* Narrowed the applied scope of metadata annotation to list/leaf-list instances

Authors’ Addresses

Qiufang Ma
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: maqiufang1@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Balazs Lengyel
Ericsson
Email: balazs.lengyel@ericsson.com

Hongwei Li
HPE
Email: flycoolman@gmail.com
Abstract

This document updates NMDA [RFC8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To avoid clients’ explicit copy/paste of referenced system-defined configuration into the target configuration datastore (e.g., <running>), a "resolve-system" parameter has been defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. The solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay and reference nodes defined in <system>, override 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 12 October 2022.
1. Introduction ........................................... 3
   1.1. Terminology ....................................... 4
   1.2. Requirements Language ........................... 5
   1.3. Updates to RFC 8342 .............................. 5
   1.4. Updates to RFC 6241, RFC 8526 ................ 5
   1.5. Updates to RFC 8040 .............................. 6
       1.5.1. Query Parameter ............................ 6
       1.5.2. Query Parameter URI ........................ 6
2. Kinds of System Configuration .......................... 7
   2.1. Immediately-Active ............................... 7
   2.2. Conditionally-Active ............................. 7
   2.3. Inactive-Until-Referenced ....................... 7
3. Static Characteristics ................................ 7
   3.1. Read-only to Clients ............................. 7
   3.2. May Change via Software Upgrades ............... 8
   3.3. No Impact to <operational> ..................... 8
4. Dynamic Behavior ....................................... 8
   4.1. Conceptual Model ................................. 8
   4.2. Explicit Declaration of System Configuration .... 9
   4.3. Servers Auto-configuring Referenced System
        Configuration ................................... 10
   4.4. Modifying (overriding) System Configuration .... 10
   4.5. Examples ......................................... 11
       4.5.1. Server Configuring of <running> Automatically ... 11
       4.5.2. Declaring a System-defined Node in <running>
              Explicitly ................................... 17
       4.5.3. Modifying a System-instantiated Leaf’s Value .... 20
       4.5.4. Configuring Descendant Nodes of a System-defined
              Node ........................................ 22
5. The <system> Configuration datastore .................. 23
6. The "ietf-system-datastore" Module .................... 25
   6.1. Data Model Overview ............................. 25
1. Introduction

NMDA [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, the client references a system configuration which isn’t present in the target datastore (e.g., `<running>`). Having to copy the entire contents of the system configuration into the target datastore should be avoided or reduced when possible while ensuring that all referential integrity constraints are satisfied.
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).

This document updates NMDA [RFC8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To avoid clients’ explicit copy/paste of referenced system-defined configuration into the target configuration datastore (e.g., <running>), a "resolve-system" parameter has been defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. The solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

Conformance to this document requires servers to implement the "ietf-system-datastore" YANG Module.

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. System configuration is present in <system> once it’s created (regardless of being applied by the device), and appears in <intended> which is subject to validation. Applied system configuration also appears in <operational> with origin="system".

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

This document redefines the term "conventional configuration datastore" from RFC 8342 to add "system" to the list of conventional configuration datastores:

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.

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 8342

This document updates RFC 8342 to define a configuration datastore called "system" to hold system configuration, it also redefines the term "conventional configuration datastore" from RFC 8342 to add "system" to the list of conventional configuration datastores. The contents of <system> datastore are read-only to clients but may change dynamically. The <system> aware client may retrieve all three types of system configuration defined in Section 2, reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

The server will merge <running> and <system> to create <intended>. As always, system configuration will appear in <operational> with origin="system" when it is in use.

The <system> datastore makes system configuration visible to clients in order for being referenced or configurable prior to present in <operational>.

1.4. Updates to RFC 6241, RFC 8526

This document augments <edit-config> and <edit-data> RPC operations defined in [RFC6241] and [RFC8526] respectively, with a new additional input parameter "resolve-system". The <copy-config> RPC operation defined in [RFC6241] is also augmented to support "resolve-system" parameter.

The "resolve-system" parameter is optional and has no value. When it is provided and the server detects that there is a reference to a system-defined node during the validation, the server will automatically copy the referenced system configuration into the validated datastore to make the configuration valid without the
client doing so explicitly. Legacy Clients interacting with servers
that support this parameter don't see any changes in <edit-
config>/<edit-data> and <copy-config> behaviors.

According to the NETCONF constraint enforcement model defined in the
section 8.3 of [RFC7950], if the target datastore of the <edit-
config>/<edit-data> or <copy-config> is "running" or "startup", the
server’s copy referenced nodes from <system> to the target datastore
MUST be enforced at the end of the <edit-config>/<edit-data> or
<copy-config> operations during the validation. If the target
datastore of the <edit-config>/<edit-data> or <copy-config> is
"candidate", the server’s copy referenced nodes from <system> to the
target datastore is delayed until a <commit> or <validate> operation
takes place.

1.5. Updates to RFC 8040

This document extends Section 4.8 and Section 9.1.1 of [RFC8040] to
add a new query parameter "resolve-system" and corresponding query
parameter capability URI.

1.5.1. Query Parameter

The "resolve-system" parameter controls whether to allow a server
copy any referenced system-defined configuration automatically
without the client doing so explicitly. 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.

+----------------+---------+-----------------------------------------+
<table>
<thead>
<tr>
<th>Name</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolve-system</td>
<td>POST,</td>
<td>resolve any references not resolved by</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>the client and copy referenced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system configuration into &lt;running&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>automatically. This parameter can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>given in any order.</td>
</tr>
</tbody>
</table>

1.5.2. Query Parameter URI

To enable the RESTCONF client to discover if the "resolve-system"
query parameter is supported by the server, the following capability
URI is defined, which is advertised by the server if supported, using
the "ietf-restconf-monitoring" module defined in RFC 8040:

urn:ietf:params:restconf:capability:resolve-system:1.0
2. Kinds of System Configuration

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

2.1. Immediately-Active

Immediately-active system configurations are those generated in <system> and applied immediately when the device is powered on (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 generated in <system> and applied 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 generated in <system> immediately when it is powered on, but they are not applied and active until being referenced.

3. Static Characteristics

3.1. Read-only to Clients

The <system> configuration datastore is a read-only configuration datastore (i.e., edits towards <system> directly MUST be denied), though the client may be allowed to override the value of a system-initialized data node (see Section 4.4). Configuration defined in <system> is merged into <intended>, and present in <operational> if it is actively in use by the device. Thus unless the resource is no longer available (e.g., the interface removed physically), there is no way to actually delete system configuration from a server, even if a client may be allowed to delete the configuration copied from
<system> into <running>. Any deletable system-provided configuration must be defined in <factory-default> [RFC8808], 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 device upgrade or if system-controlled resources (e.g., HW available) change. In some implementations, when QoS function is enabled, QoS-related 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. Any update of the contents in <system> will not cause the automatic update of <running>, even if some of the system configuration has already been copied into <running> explicitly or automatically before the update.

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 out of scope, hence existing "config false" nodes are not impacted by this work.

4. Dynamic Behavior

4.1. Conceptual Model

This document introduces a mandatory 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-active system configuration will be generated in <system> and applied immediately but inactive-until-referenced system configuration only becomes active if it is referenced by client-defined configuration. While conditionally-active system configuration will be created and immediately applied if the condition on system resources is met when the device is powered on or running.
All above three types of system configurations will appear in <system>. Clients MAY reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes, by copying or writing intended configurations into the target configuration datastore (e.g., <running>).

The server will merge <running> and <system> to create <intended>, in which process, the data node appears in <running> if the server allows the node to be modifiable; additional nodes to a list entry or new list/leaf-list entries appear in <running> extends the list entry or the whole list/leaf-list defined in <system> if the server allows the list/leaf-list to be updated. In addition, the <intended> configuration datastore represents the configuration after all configuration transformation to <system> are performed (e.g., system-defined template expansion, removal of inactive system configuration). If a server implements <intended>, <system> MUST be merged into <intended>.

Servers MUST enforce that configuration references in <running> are resolved within the <running> datastore and ensure that <running> contains any referenced system objects. Clients MUST either explicitly copy system-defined nodes into <running> or use the "resolve-system" parameter. The server MUST enforce that the referenced system nodes configured into <running> by the client is consistent with <system>. Note that <system> aware clients know how to discover what nodes exist in <system>. How clients unaware of the <system> datastore can find appropriate configurations is beyond the scope of this document.

No matter how the referenced system objects are copied into <running>, the nodes copied into <running> would always be returned after a read of <running>, regardless if the client is <system> aware.

4.2. Explicit Declaration of System Configuration

It is possible for a client to explicitly declare system configuration nodes in the target datastore (e.g., <running>) with the same values as in <system>, by configuring a node (list/leaf-list entry, leaf, etc) in the target datastore (e.g., <running>) that matches the same node and value in <system>.

This explicit configuration of system-defined nodes in <running> can be useful, for example, when the client doesn't want a "system client" to have a role or hasn't implemented the "resolve-system" parameter. The client can explicitly declare (i.e. configure in <running>) the list entries (with at least the keys) for any system

Ma, et al. Expires 12 October 2022 [Page 9]
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.

4.3. Servers Auto-configuring Referenced System Configuration

This document defines a new parameter "resolve-system" to the input for the <edit-config>, <edit-data> and <copy-config> operations. Clients that are aware of the "resolve-system" parameter MAY use this parameter to avoid the requirement to provide a referentially complete configuration in <running>.

If the "resolve-system" is present, the server MUST copy relevant referenced system-defined nodes into the target datastore (e.g., <running>) without the client doing the copy/paste explicitly, to resolve any references not resolved by the client. The server acting as a "system client" like any other remote clients copies the referenced system-defined nodes when triggered by the "resolve-system" parameter. If the "resolve-system" parameter is not given by the client, the server SHOULD NOT modify <running> in any way otherwise not specified by the client.

The server may automatically configure the list entries (with at least the keys) in the target datastore (e.g., <running>) for any system configuration list entries that are referenced elsewhere by the clients. Similarly, not all the contents of the list entry (i.e., the descendant nodes) are necessarily copied by the server - only the parts that are required to make the <running> valid. A read back of <running> (i.e., <get>, <get-config> or <get-data> operation) returns those automatically copied nodes.

4.4. 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. Even if some system configuration has been copied into <running> earlier, whether it is modifiable or not in <running> follows general YANG and NACM rules, and other server-internal restrictions. If a system configuration node is non-modifiable, then writing a different value for that node in <running> MUST return an error. The immutability of system configuration is further defined in [I-D.ma-netmod-immutable-flag].
Modification of system configuration is achieved by the client writing configuration to `<running>` that overrides the system configuration. Configurations defined in `<running>` take precedence over system configuration nodes in `<system>` if the server allows the nodes to be modified.

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 unless the resource is no longer available. For example, a "mandatory true" leaf 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. Another example of this might be that system initializes a value for a particular leaf which is overridden by the client with intended value in `<running>`. The client may delete the leaf in `<running>`, but system-initialized value defined in `<system>` will be in use and appear in `<operational>`.

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.

4.5. Examples

This section shows the examples of server-configuring of `<running>` automatically, declaring a system-defined node in `<running>` explicitly, modifying a system-instantiated leaf’s value and configuring descendant nodes of a system-defined node. For each example, the corresponding XML snippets are provided.

4.5.1. Server Configuring of `<running>` Automatically

In this subsection, the following fictional module is used:
module example-application {
  yang-version 1.1;
  namespace "urn:example:application";
  prefix "app";

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

  container applications {
    list application {
      key "name";
      leaf name {
        type string;
      }
      leaf protocol {
        type enumeration {
          enum tcp;
          enum udp;
        }
      }
      leaf destination-port {
        type inet:port-number;
      }
    }
  }
}

The server may predefine some applications as a convenience for the clients. These predefined objects are applied only after being referenced by other configurations, which fall into the "inactive-until-referenced" system configuration as defined in Section 2. The system-instantiated application entries may be present in <system> as follows:
<applications xmlns="urn:example:application">
  <application>
    <name>ftp</name>
    <protocol>tcp</protocol>
    <destination-port>21</destination-port>
  </application>
  <application>
    <name>tftp</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application>
    <name>smtp</name>
    <protocol>tcp</protocol>
    <destination-port>25</destination-port>
  </application>
  ...
</applications>

The client may also define its customized applications. Suppose the configuration of applications is present in <running> as follows:

<applications xmlns="urn:example:application">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
</applications>

A fictional ACL YANG module is used as follows, which defines a leafref for the leaf-list "application" data node to refer to an existing application name.
module example-acl {
  yang-version 1.1;
  namespace "urn:example:acl";
  prefix "acl";

  import example-application {
    prefix "app";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  container acl {
    list acl_rule {
      key "name";
      leaf name {
        type string;
      }
      container matches {
        choice l3 {
          container ipv4 {
            leaf source_address {
              type inet:ipv4-prefix;
            }
            leaf destination_address {
              type inet:ipv4-prefix;
            }
          }
        }
        choice applications {
          leaf-list application {
            type leafref {
              path "/app:applications/app:application/app:name";
            }
          }
        }
      }
      leaf packet_action {
        type enumeration {
          enum forward;
          enum drop;
          enum redirect;
        }
      }
    }
  }
}
If a client configures an ACL rule referencing system predefined nodes which are not present in <running>, the client MAY issue an <edit-config> operation with the parameter "resolve-system" as follows:

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <acl xmlns="urn:example:acl">
        <acl_rule>
          <name>allow_access_to_ftp_tftp</name>
          <matches>
            <ipv4>
              <source_address>198.51.100.0/24</source_address>
              <destination_address>192.0.2.0/24</destination_address>
            </ipv4>
            <application>ftp</application>
            <application>tftp</application>
            <application>my-app-1</application>
          </matches>
          <packet_action>forward</packet_action>
        </acl_rule>
      </acl>
    </config>
  </edit-config>
</rpc>
```

Then following gives the configuration of applications in <running> which is returned in the response to a follow-up <get-config> operation:
<applications xmlns="urn:example:application">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application>
    <name>ftp</name>
    <protocol>tcp</protocol>
  </application>
  <application>
    <name>tftp</name>
  </application>
</applications>

Then the configuration of applications is present in <operational> as follows:

<applications xmlns="urn:example:application"
    xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
    or:origin="or:intended">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application or:origin="or:system">
    <name>ftp</name>
    <protocol>tcp</protocol>
    <destination-port>21</destination-port>
  </application>
  <application or:origin="or:system">
    <name>tftp</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
</applications>
Since the configuration of application "smtp" is not referenced by the client, it does not appear in <operational> but only in <system>.

4.5.2. Declaring a System-defined Node in <running> Explicitly

It’s also possible for a client to explicitly declare the system-defined configurations that are referenced. For instance, in the above example, the client MAY also explicitly configure the following system defined applications "ftp" and "tftp" only with the list key "name" before referencing:

```
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
   <target>
     <running/>
   </target>
   <config>
     <applications xmlns="urn:example:application">
       <application>
         <name>ftp</name>
       </application>
       <application>
         <name>tftp</name>
       </application>
     </applications>
   </config>
 </edit-config>
</rpc>
```

Then the client issues an <edit-config> operation to configure an ACL rule referencing applications "ftp" and "tftp" without the parameter "resolve-system" as follows:
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config>
            <acl xmlns="urn:example:acl">
                <acl_rule>
                    <name>allow_access_to_ftp_tftp</name>
                    <matches>
                        <ipv4>
                            <source_address>198.51.100.0/24</source_address>
                            <destination_address>192.0.2.0/24</destination_address>
                        </ipv4>
                        <application>ftp</application>
                        <application>tftp</application>
                        <application>my-app-1</application>
                    </matches>
                    <packet_action>forward</packet_action>
                </acl_rule>
            </acl>
        </config>
    </edit-config>
</rpc>

Then following gives the configuration of applications in <running>
which is returned in the response to a follow-up <get-config>
operation, all the configuration of applications are explicitly
configured by the client:
Then the configuration of applications is present in <operational> as follows:

```xml
<applications xmlns="urn:example:application"
    xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
    or:origin="or:intended"/>

<application>
  <name>my-app-1</name>
  <protocol>tcp</protocol>
  <destination-port>2345</destination-port>
</application>

<application>
  <name>my-app-2</name>
  <protocol>udp</protocol>
  <destination-port>69</destination-port>
</application>

<application>
  <name>ftp</name>
  <protocol>tcp</protocol>
  <destination-port>21</destination-port>
</application>

<application>
  <name>tftp</name>
  <protocol>udp</protocol>
  <destination-port>69</destination-port>
</application>
```

Ma, et al. Expires 12 October 2022 [Page 19]
Since the application names "ftp" and "tftp" are explicitly configured by the client, they take precedence as the value in <system>, the "origin" attribute will be set to "intended".

4.5.3. Modifying a System-instantiated Leaf’s Value

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

```yang
module example-qos-policy {
  yang-version 1.1;
  namespace "urn:example:qos";
  prefix "qos";

  container qos-policies {
    list policy {
      key "name";
      leaf name {
        type string;
      }
    }
    list queue {
      key "queue-id";
      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 xmlns="urn:example:qos">
  <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 xmlns="urn:example:qos">
        <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="urn:example:qos"
  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>

4.5.4. Configuring Descendant Nodes of a System-defined Node

This subsection also uses the fictional interface YANG module defined in Appendix C.3 of [RFC8342]. 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:
Later on, the client further configures the description node of a "lo0" interface as follows:

```xml
<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:

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

5. The <system> Configuration Datastore

NMDA servers claiming to support this document MUST implement a <system> configuration datastore, and they SHOULD also implement the <intended> datastore.
Following guidelines for defining datastores in the appendix A of [RFC8342], this document introduces a new 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 up to the root of the tree, generated by the system

* 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 flows into <operational>. The "system" origin Metadata Annotation [RFC7952] is used to indicate the origin of a data item is system.

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

* Defining YANG module: "ietf-system-datastore".

The datastore’s content is defined 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 device upgrade or system-controlled resources change (e.g., HW available). The <system> datastore doesn’t persist across reboots; the contents of <system> will be lost upon reboot and recreated by the system with the same or changed contents. <factory-reset> RPC operation defined in [RFC8808] can reset it to its factory default configuration without including configuration generated due to the system update or client-enabled functionality.

The <system> datastore is defined as a conventional configuration datastore and shares a common datastore schema with other conventional datastores. The <system> configuration datastore must always be valid, as defined in Section 8.1 of [RFC7950].
6. The "ietf-system-datastore" Module

6.1. Data Model Overview

This YANG module defines a new YANG identity named "system" that uses the "ds:datastore" identity defined in [RFC8342]. A client can discover the <system> datastore support on the server by reading the YANG library information from the operational state datastore. 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 is system. Support for the "origin" annotation is identified with the feature "origin" defined in [RFC8526].

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].

6.2. Example Usage

This section gives an example of data retrieval from <system>. The YANG module used are shown in Appendix C.2 of [RFC8342]. All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

Suppose the following data is added to <running>:

```json
{
    "bgp": {
        "local-as": "64501",
        "peer-as": "64502",
        "peer": {
            "name": "2001:db8::2:3"
        }
    }
}
```
REQUEST (a <get-data> or GET request sent from the NETCONF or RESTCONF client):

Datastore: <system>
Target:/bgp

An example of RESTCONF request:

GET /restconf/ds/system/bgp HTTP/1.1
Host: example.com
Accept: application/yang-data+xml

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

{  
  "bgp": {  
    "peer": {  
      "name": "2001:db8::2:3",  
      "local-port": "60794"  
    }  
  }  
}

6.3. 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>
    Author: Qiufang Ma
      <mailto:maqiufang1@huawei.com>
    Author: Chong Feng
      <mailto:frank.fengchong@huawei.com>
This module defines a new YANG identity that uses the ds:datastore identity defined in [RFC8342].

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 HHHH (https://www.rfc-editor.org/info/rfcHHHH); 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."

revision 2021-05-14 {
  description
    "Initial version.";
  reference
    "RFC XXXX: System-defined Configuration";
}

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

<CODE ENDS>
7. The "ietf-netconf-resolve-system" Module

This YANG module is optional to implement.

7.1. Data Model Overview

This YANG module augments NETCONF <edit-config>, <edit-data> and 
<copy-config> operations with a new parameter "resolve-system" in the 
input parameters. If the "resolve-system" parameter is present, the 
server will copy the referenced system configuration into target 
datastore automatically. A NETCONF client can discover the "resolve-

system" parameter support on the server by checking the YANG library 
information with "ietf-netconf-resolve-system" included from the 
operational state datastore.

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

module: ietf-netconf-resolve-system
  augment /nc:edit-config/nc:input:
      +---w resolve-system?   empty
  augment /nc:copy-config/nc:input:
      +---w resolve-system?   empty
  augment /ncds:edit-data/ncds:input:
      +---w resolve-system?   empty

The following tree diagram [RFC8340] illustrates "edit-config", 
"copy-config" and "edit-data" rpcs defined in "ietf-netconf" and 
"ietf-netconf-nmda" respectively, augmented by "ietf-netconf-resolve-
system" YANG module :

rpcs:
  +---x edit-config
      +---w input
          +---w target
              +---w (config-target)
                  +--:(candidate)
                      | | +---w candidate?   empty {candidate}?
                      | +--:(running)
                          | +---w running?   empty {writable-running}?
              +---w default-operation?   enumeration
              +---w test-option?   enumeration {validate}?
              +---w error-option?   enumeration
              +---w (edit-content)
                  +--:(config)
                      | | +---w config?   <anyxml>
                      | +--:(url)
                          | +---w url?   inet:uri {url}?
7.2. Example Usage

This section gives an example of an <edit-config> request to reference system-defined data nodes which are not present in <running> with a "resolve-system" parameter. A retrieval of <running> to show the auto-copied referenced system objects after the <edit-config> request is also given. The YANG module used is shown as follows, leafrefs refer to an existing name and address of an interface:
module example-interface-management {
    yang-version 1.1;
    namespace "urn:example:interfacemgmt";
    prefix "inm";

    container interfaces {
        list interface {
            key name;
            leaf name {
                type string;
            }
            leaf description {
                type string;
            }
            leaf mtu {
                type uint16;
            }
            leaf ip-address {
                type inet:ip-address;
            }
        }
    }

    container default-address {
        leaf ifname {
            type leafref {
                path "../../interfaces/interface/name";
            }
        }
        leaf address {
            type leafref {
                path "../../interfaces/interface[name = current()]/../ifname" + "./ip-address";
            }
        }
    }
}

Image that the system provides a loopback interface (named "lo0")
with a predefined MTU value of "1500" and a predefined IP address of
"127.0.0.1". The <system> datastore shows the following
configuration of loopback interface:
The client sends an <edit-config> operation to add the configuration of default-address with a "resolve-system" parameter:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <default-address xmlns="urn:example:interfacemgmt">
        <if-name>lo0</if-name>
        <address>127.0.0.1</address>
      </default-address>
    </config>
    <resolve-system/>
  </edit-config>
</rpc>
```

Since the "resolve-system" parameter is provided, the server will resolve any leafrefs to system configurations and copy the referenced system-defined nodes into <running> automatically with the same value (i.e., the name and ip-address data nodes of lo0 interface) in <system> at the end of <edit-config> operation constraint enforcement. After the processing, a positive response is returned:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Then the client sends a <get-config> operation towards <running>:
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <interfaces xmlns="urn:example:interfacemgmt"/>
    </filter>
  </get-config>
</rpc>

Given that the referenced interface "name" and "ip-address" of lo0 are configured by the server, the following response is returned:

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="urn:example:interfacemgmt">
      <interface>
        <name>lo0</name>
        <ip-address>127.0.0.1</ip-address>
      </interface>
    </interfaces>
  </data>
</rpc-reply>

7.3. YANG Module

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

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

  import ietf-netconf-nmda {
    prefix ncds;
    reference
      "RFC 8526: NETCONF Extensions to Support the Network Management Datastore Architecture";
  }
}
</CODE BEGINS>
This module defines an extension to the NETCONF protocol that allows the NETCONF client to control whether the server is allowed to copy referenced system configuration automatically without the client doing so explicitly.

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 HHHH (https://www.rfc-editor.org/info/rfcHHHH); 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.

revision 2021-05-14 {
  description
    "Initial version.";
  reference
    "RFC XXXX: System-defined Configuration";
}

augment /nc:edit-config/nc:input {

Ma, et al. Expires 12 October 2022 [Page 33]
description
"Allows the server to automatically configure referenced system configuration to make configuration valid.";
leaf resolve-system {
  type empty;
  description
  "When present, the server is allowed to automatically configure referenced system configuration into the target configuration datastore.";
}
}

augment /nc:copy-config/nc:input {
  description
  "Allows the server to automatically configure referenced system configuration to make configuration valid.";
  leaf resolve-system {
    type empty;
    description
    "When present, the server is allowed to automatically configure referenced system configuration into the target configuration datastore.";
  }
}

augment /ncds:edit-data/ncds:input {
  description
  "Allows the server to automatically configure referenced system configuration to make configuration valid.";
  leaf resolve-system {
    type empty;
    description
    "When present, the server is allowed to automatically configure referenced system configuration into the target configuration datastore.";
  }
}

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].

Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

8.2. The "YANG Module Names" Registry

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

name: ietf-system-datastore
prefix: sys
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

name: ietf-netconf-resolve-system
prefix: ncrs
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

8.3. RESTCONF Capability URN Registry

This document registers a capability in the "RESTCONF Capability URNs" registry [RFC8040]:

<table>
<thead>
<tr>
<th>Index</th>
<th>Capability Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>:resolve-system</td>
<td>urn:ietf:params:restconf:capability:resolve-system:1.0</td>
</tr>
</tbody>
</table>

9. Security Considerations

9.1. 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.

9.2. Regarding the "ietf-netconf-resolve-system" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and [RFC8526]. 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.

10. Contributors

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf@chinatelecom.cn

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Acknowledgements

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

References

Normative References

Ma, et al. Expires 12 October 2022 [Page 36]
Informative References

[I-D.ma-netmod-immutable-flag]


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>
     <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>
       <description>Test interface</description>
     </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>
  <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>
    <description>Test interface</description>
    <mtu>1500</mtu>
  </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>
</interfaces>

Appendix B. Changes between Revisions

v02 - v03

* Define a RESTCONF capability URI for "resolve-system" RESTCONF query parameter;

* Augment <copy-config> RPC operation to support "resolve-system" for input parameter;

* Editorial changes for clarification and explanation. E.g., definition of system configuration, is <system> always valid? Will the update of <system> be reflected into <running>? Clarify "read-only to clients" and "modifying system configuration", non-deletable system configuration, etc

v00 - v02

* Remove the "with-system" parameter to retrieve <running> with system configuration merged in.

* Add a new parameter named "resolve-system" to allow the server to populate referenced system configuration into <running> automatically in order to make <running> valid.

* Usage examples refinement.

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

* Should the "with-origin" parameter be supported for <intended>?

Authors’ Addresses

Qiufang Ma (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: maqiufang1@huawei.com

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net