Network Modeling
Internet-Draft
Obsoletes: 8407 (if approved)
Updates: 8126 (if approved)
Intended status: Best Current Practice
Expires: 7 January 2024

M. Boucadair
Orange
Q. Wu
Huawei

6 July 2023

Guidelines for Authors and Reviewers of Documents Containing YANG Data Models

draft-boucadair-netmod-rfc8407bis-01

Abstract

This memo provides guidelines for authors and reviewers of specifications containing YANG modules, including IANA-maintained modules. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) and RESTCONF protocol implementations that utilize YANG modules. This document obsoletes RFC 8407.

Also, this document updates RFC 8126 by providing additional guidelines for writing the IANA considerations for RFCs that specify IANA-maintained modules.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Network Modeling Working Group mailing list (netmod@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netmod/.

Source for this draft and an issue tracker can be found at https://github.com/boucadair/rfc8407bis.

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/.

Copyright Notice

Table of Contents

1. Introduction

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

This document defines a set of usage guidelines for documents containing YANG 1.1 [RFC7950] and YANG 1.0 [RFC6020] data models, including IANA-maintained modules. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF and/or RESTCONF server. YANG is also used to define abstract data structures [RFC8791]. A NETCONF or RESTCONF server that supports a particular YANG module will support client NETCONF and/or RESTCONF operation requests, as indicated by the specific content defined in the YANG module.

Many YANG constructs are defined as optional to use, such as the "description" statement. However, in order to make YANG modules more useful, it is desirable to define a set of usage guidelines that entails a higher level of compliance than the minimum level defined in the YANG specification [RFC7950].

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

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

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

Section 5.3 updates [RFC8126] by providing guidance for writing the IANA considerations for RFCs that specify IANA-maintained modules.

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

1.1. Changes since RFC 8407

The following changes have been made to the guidelines published in [RFC8407]:

* Implemented errata 5693, 5800, 6899, and 7416.

* Updated the terminology.

* Updated the URL of the IETF authors guidelines.

* Added code markers for the security template.

* Updated the YANG security considerations template to reflect the latest version maintained in the Wiki.

* Added statements that the security template is not required for modules that follow [RFC8791].

* Added a statement that the RFCs that are listed in the security template are to be listed as normative references in documents that use the template.

* Added a note that folding of the examples should be done as per [RFC8792] conventions.

* Added a note that RFC8792-folding of YANG modules can be used if and only if native YANG features (e.g., break line, "+") are not sufficient.

* Added tool validation checks to ensure that YANG modules fit into the line limits of an I-D.

* Added tool validation checks of JSON-encoded examples.

* Updated many examples to be aligned with the consistent indentation recommendation.

* Updated the IANA considerations to encourage registration requests to indicate whether a module is maintained by IANA or not.

* Added guidelines for IANA-maintained modules.

2. Terminology

The following terms are used throughout this document:

IANA-maintained module: A YANG module that is maintained by IANA (e.g., "iana-tunnel-type" [RFC8675] or "iana-pseudowire-types" [RFC9291]).

IETF module: A YANG module that is published by the IETF and which is not maintained by IANA.

published: A stable release of a module or submodule. For example, the "Request for Comments" described in Section 2.1 of [RFC2026] is considered a stable publication.

unpublished: An unstable release of a module or submodule. For example the "Internet-Draft" described in Section 2.2 of [RFC2026] is considered an unstable publication that is a work in progress, subject to change at any time.

YANG fragment: A set of YANG statements that are not intended to

represent a complete YANG module or submodule. These statements are not intended for actual use, except to provide an example of YANG statement usage. The invalid syntax "..." is sometimes used to indicate that additional YANG statements would be present in a real YANG module.

YANG tree diagram: A diagram representing the contents of a YANG module, as defined in [RFC8340]. It is also called a "tree diagram".

2.1. NETCONF Terms

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

* capabilities
* client
* operation
* server

2.2. YANG Terms

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

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

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

2.3. NMDA Terms

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

* configuration
* conventional configuration datastore
* datastore
* operational state
* operational state datastore

2.4. Requirements Notation

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

3. General Documentation Guidelines

YANG modules under review are likely to be contained in Internet-Drafts (I-Ds). All guidelines for I-D authors [ID-Guidelines] MUST be followed. The guidelines for RFCs should be followed and are defined in the following: [RFC7322] (and any future RFCs that obsolete it), [RFC-STYLE], and [RFC7841].

The following sections MUST be present in an I-D containing a YANG module:

* Narrative sections
* Definition sections
* Security Considerations section
* IANA Considerations section
* References section

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

* normative module or submodule

* example module or submodule

* example YANG fragment not part of any module or submodule

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

3.1. Module Copyright

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

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

3.2. Code Components

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

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

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

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

3.2.1. Example Modules

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

An example module SHOULD be named using the term "example", followed by a hyphen, followed by a descriptive name, e.g., "example-toaster".

See Section 4.9 regarding the namespace guidelines for example modules.

3.3. Terminology Section

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

3.4. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module and SHOULD be included to help readers understand YANG module structure. Guidelines on tree diagrams can be found in Section 3 of [RFC8340].

If YANG tree diagrams are used, then an informative reference to the YANG tree diagrams specification MUST be included in the document. Refer to Section 2.2 of [RFC8349] for an example of such a reference.

3.5. Narrative Sections

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

If the module or modules defined by the specification imports definitions from other modules (except for those defined in [RFC7950] or [RFC6991]) or are always implemented in conjunction with other modules, then those facts MUST be noted in the overview section; any special interpretations of definitions in other modules MUST be noted as well. Refer to Section 2.3 of [RFC8349] for an example of this overview section.

If the document contains a YANG module(s) that is compliant with NMDA [RFC8342], then the Introduction section should mention this fact.

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

Consistent indentation SHOULD be used for all examples, including YANG fragments and protocol message instance data. If line wrapping is done for formatting purposes, then this SHOULD be noted following [RFC8792], as shown in the following example:

```
============== NOTE: '\' line wrapping per RFC 8792 ================

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

Native YANG features (e.g., breaking line, "+") SHOULD be used to fit a module into the line limits. Exceptionally, RFC8792-folding of YANG modules MAY be used if and only if native YANG features are not sufficient. A similar approach (e.g., use "--yang-line-length 69" or split a tree into subtrees) SHOULD be followed for tree diagrams.

3.6. Definitions Section

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

A YIN syntax version of the module MAY also be present in the document. There MAY also be other types of modules present in the document, such as Structure of Management Information Version 2 (SMIv2), which are not affected by these guidelines.

Note that if the module itself is considered normative and not an example module or example YANG fragment, then all YANG statements within a YANG module are considered normative. The use of keywords defined in [RFC2119] and [RFC8174] apply to YANG "description" statements in normative modules exactly as they would in any other normative section.

Example YANG modules and example YANG fragments MUST NOT contain any normative text, including any all-uppercase reserved words from [RFC2119] and [RFC8174].

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

See Section 4 for guidelines on YANG usage.

3.7. Security Considerations Section

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

Unless the modules comply with [RFC8791], the security section MUST be patterned after the latest approved template (available at <https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines>). Section 3.7.1 contains the security considerations template dated 2013-05-08 and last updated on 2018-10-18. Authors MUST check the web page at the URL listed above in case there is a more recent version available.

In particular:

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

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

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

Documents that define exclusively modules following the extension in [RFC8791] are not required to include the security template in Section 3.7.1.

3.7.1. Security Considerations Section Template

```
<CODE BEGINS>
X. Security Considerations

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

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

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

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

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

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

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

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

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

<list RPC operations and state why they are sensitive>
<CODE ENDS>

Note: [RFC8446], [RFC6241], [RFC6242], [RFC8341], and [RFC8040] (or

future RFCs that replace any of them) have to be listed as normative references.

3.8. IANA Considerations Section

In order to comply with IESG policy as set forth in <https://www.ietf.org/id-info/checklist.html>, every I-D that is submitted to the IESG for publication MUST contain an IANA Considerations section. The requirements for this section vary depending on what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section will state that "This document has no IANA actions". Refer to the guidelines in [RFC8126] for more details.

Each normative YANG module MUST be registered in both the "IETF XML Registry" [RFC3688] [IANA-XML] and the "YANG Module Names" registry [RFC6020] [IANA-MOD-NAMES]. The registration request in the "YANG Module Names" registry should indicate whether the module is IANA-maintained or not. This applies to new modules and updated modules. An example of an update registration for the "ietf-template" module can be found in Section 6.

Additional IANA considerations applicable to IANA-maintained modules are provided in Section 5.3.

3.8.1. Documents That Create a New Namespace

If an I-D defines a new namespace that is to be administered by the IANA, then the document MUST include an IANA Considerations section that specifies how the namespace is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new entry in the "ns" subregistry within the "IETF XML Registry" MUST be requested from the IANA.

3.8.2. Documents That Extend an Existing Namespace

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

3.9. References Sections

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

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

3.10. Validation Tools

All modules need to be validated before submission in an I-D. The 'pyang' YANG compiler is freely available from GitHub:

<https://github.com/mbj4668/pyang>

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

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

To ensure that a module fits into the line limits of an I-D, the command "pyang -f yang --keep-comments --yang-line-length 69" should be used.

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

<https://github.com/CESNET/libyang>

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

To check that JSON-encoded examples [RFC7951] comply with the target data models, "yangson" program should be used. The "yangson" program is freely available from GitHub.

<https://github.com/CZ-NIC/yangson>

3.11. Module Extraction Tools

A version of 'rfcstrip' that will extract YANG modules from an I-D or RFC is available. The 'rfcstrip' tool that supports YANG module extraction is freely available at:

<https://github.com/mbj4668/rfcstrip>

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

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

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

3.12. Module Usage Examples

Each specification that defines one or more modules SHOULD contain usage examples, either throughout the document or in an appendix. This includes example instance document snippets in an appropriate encoding (e.g., XML and/or JSON) to demonstrate the intended usage of the YANG module(s). Example modules MUST be validated. Refer to Section 3.10 for tools that validate YANG modules and examples. If IP addresses are used, then a mix of either IPv4 and IPv6 addresses or IPv6 addresses exclusively SHOULD be used in the examples. IPv4 and IPv6 addresses/prefixes reserved for documentation are defined [RFC5737] and [RFC3849].

4. YANG Usage Guidelines

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

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

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

4.1. Module Naming Conventions

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

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

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

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

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

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

4.2. Prefixes

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

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

  container foo;
}

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

  typedef foo { type uint32; }
}

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

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

YANG defines the following rules for prefix usage:

* Prefixes are never used for built-in data types and YANG keywords.

* A prefix MUST be used for any external statement (i.e., a statement defined with the YANG "extension" statement).

* The proper module prefix MUST be used for all identifiers imported from other modules.

* The proper module prefix MUST be used for all identifiers included from a submodule.

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

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

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

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

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

4.3. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an "identifier-arg-str" token in the ABNF in Section 14 of [RFC7950].

4.3.1. Identifier Naming Conventions

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

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

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

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

4.4. Defaults

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

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

Table 1: Statement Defaults

4.5. Conditional Statements

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

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

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

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

If any "if-feature" statements apply to a list node, then the same "if-feature" statements MUST apply to any key leaf nodes for the list. There MUST NOT be any "if-feature" statements applied to any key leafs that do not also apply to the parent list node.

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

4.6. XPath Usage

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

4.6.1. XPath Evaluation Contexts

YANG defines five separate contexts for evaluation of XPath statements:

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

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

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

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

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

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

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

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

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

4.6.2. Function Library

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

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

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

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

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

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

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

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

Example:

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

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

4.6.3. Axes

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

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

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

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

4.6.4. Types

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

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

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

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

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

Example:

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

4.6.5. Wildcards

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

Wildcard expansion is done within a server against all the nodes from all namespaces, so it is possible for a "must" or "when" expression that uses the '*' operator to always evaluate to false if processed within a single YANG module. In such cases, the "description" statement SHOULD clarify that augmenting objects are expected to match the wildcard expansion.

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

4.6.6. Boolean Expressions

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

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

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

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

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

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

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

4.7. YANG Definition Lifecycle Management

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

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

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

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

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

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

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

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

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

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

4.8. Module Header, Meta, and Revision Statements

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

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

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

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

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

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

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

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

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

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

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

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

4.9. Namespace Assignments

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

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

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

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

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

A standard namespace statement value SHOULD have the following form:

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

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

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

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

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

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

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

Note that a different URN prefix string SHOULD be used for modules that are not Standards Track. The string SHOULD be selected according to the guidelines in [RFC7950].

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

Example URIs using URLs per [RFC3986]:

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

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

Example URIs using tags per [RFC4151]:

```
tag:example.com,2017:example-interfaces

tag:example.com,2017:example-system
```

4.10. Top-Level Data Definitions

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

The separation of configuration data and operational state SHOULD be considered carefully. It is sometimes useful to define separate top-level containers for configuration and non-configuration data. For some existing top-level data nodes, configuration data was not in scope, so only one container representing operational state was created. Refer to NMDA [RFC8342] for details.

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

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

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

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

4.11. Data Types

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

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

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

4.11.1. Fixed-Value Extensibility

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

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

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

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

identity one {
  base f:foo-type;
}

identity two {
  base f:foo-type;
}

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

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

4.11.2. Patterns and Ranges

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

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

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

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

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

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

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

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

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

4.11.3. Enumerations and Bits

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

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

4.11.4. Union Types

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

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

Incorrect:

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

Correct:

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

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

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

Incorrect:

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

Correct:

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

4.11.5. Empty and Boolean

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

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

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

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

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

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

Incorrect:

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

Correct:

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

4.12. Reusable Type Definitions

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

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

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

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

The "description" statement MUST be present.

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

4.13. Reusable Groupings

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

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

* Clearly identify the purpose of the grouping in the "description" statement.

* There are five different XPath contexts in YANG (rpc/input, rpc/output, notification, "config true" data nodes, and all data nodes). Clearly identify which XPath contexts are applicable or excluded for the grouping.

* Do not reference data outside the grouping in any "path", "must", or "when" statements.

* Do not include a "default" substatement on a leaf or choice unless the value applies on all possible contexts.

* Do not include a "config" substatement on a data node unless the value applies on all possible contexts.

* Clearly identify any external dependencies in the grouping "description" statement, such as nodes referenced by an absolute path from a "path", "must", or "when" statement.

4.14. Data Definitions

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

* anyxml

* augment

* choice

* container

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

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

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

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

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

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

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

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

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

4.14.1. Non-Presence Containers

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

Example using container wrappers:

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

Example without container wrappers:

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

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

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

4.14.2. Top-Level Data Nodes

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

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

4.15. Operation Definitions

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

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

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

4.16. Notification Definitions

The "description" statement MUST be present.

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

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

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

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

4.17. Feature Definitions

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

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

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

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

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

For example, feature2 requires implementation of feature1:

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

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

4.18. YANG Data Node Constraints

4.18.1. Controlling Quantity

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

4.18.2. "must" versus "when"

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

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

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

4.19. "augment" Statements

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

A top-level "augment" statement SHOULD NOT be used if the target data node is in the same module or submodule as the evaluated "augment" statement. The data definition statements SHOULD be added inline instead.

4.19.1. Conditional Augment Statements

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

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

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

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

4.19.2. Conditionally Mandatory Data Definition Statements

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

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

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

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

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

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

```
  module example-module {

    yang-version 1.1;
    namespace "tag:example.com,2017:example-module";
    prefix mymod;

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

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

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

      leaf mandatory-leaf {
        type string;
        mandatory true;
      }
    }
  }
```

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

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

4.20. Deviation Statements

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

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

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

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

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

Example documenting platform resource limits:

Wrong: (max-elements in the list itself)

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

Correct: (max-elements in a deviation)

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

4.21. Extension Statements

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

The following guidelines apply to the usage of YANG extensions:

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

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

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

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

4.22. Data Correlation

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

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

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

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

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

The relationship between configuration and operational state has been clarified in NMDA [RFC8342].

4.22.1. Use of "leafref" for Key Correlation

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

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

Not preferred:

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

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

Preferred:

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

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

4.23. Operational State

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

Operational state is now modeled using YANG according to the new NMDA [RFC8342] and conceptually contained in the operational state datastore, which also includes the operational values of configuration data. There is no longer any need to duplicate data structures to provide separate configuration and operational state sections.

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

4.23.1. Combining Operational State and Configuration Data

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

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

4.23.2. Representing Operational Values of Configuration Data

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

Sometimes the configured value set is different than the operational value set for that object, for example, the "admin-status" and "oper-status" leafs in [RFC8343]. In this case, a separate object MAY be used to represent the configured and operational values.

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

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

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

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

4.23.3. NMDA Transition Guidelines

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

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

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

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

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

(b) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] has been restructured as an NMDA-compatible model in [RFC8343]. The "/interfaces-state" hierarchy has been marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

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

4.23.3.1. Temporary Non-NMDA Modules

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

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

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

* Change the module name by appending "-state" to the original module name

* Change the namespace by appending "-state" to the original namespace value

* Change the prefix by appending "-s" to the original prefix value

* Add an import to the original module (e.g., for typedef definitions)

* Retain or create only the top-level nodes that have a "config" statement value "false". These subtrees represent "config false" data nodes that were combined into the configuration subtree; therefore, they are not available to non-NMDA aware clients. Set the "status" statement to "deprecated" for each new node.

* The module description SHOULD clearly identify the module as a temporary non-NMDA module

4.23.3.2. Example: Create a New NMDA Module

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

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

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

4.23.3.3. Example: Convert an Old Non-NMDA Module

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

Old Module:

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

    container foo {
      // configuration data child nodes
    }

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

Converted NMDA Module:

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

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

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

4.23.3.4. Example: Create a Temporary NMDA Module

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

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

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

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

4.24. Performance Considerations

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

* Lists are generally more expensive than containers

* "when" statement evaluation is generally more expensive than "if-feature" or "choice" statements

* "must" statements are generally more expensive than "min-entries", "max-entries", "mandatory", or "unique" statements

* "identityref" leafs are generally more expensive than "enumeration" leafs

* "leafref" and "instance-identifier" types with "require-instance" set to true are generally more expensive than if "require-instance" is set to false

4.25. Open Systems Considerations

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

4.26. Guidelines for Constructs Specific to YANG 1.1

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

4.26.1. Importing Multiple Revisions

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

4.26.2. Using Feature Logic

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

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

Features generally require less server implementation complexity and runtime resources than objects that use "when" statements. Features are generally static (i.e., set when a module is loaded and not changed at runtime). However, every client edit might cause a "when" statement result to change.

4.26.3. "anyxml" versus "anydata"

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

4.26.4. "action" versus "rpc"

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

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

The NETCONF Access Control Model (NACM) [RFC8341] does not support parameter-based access control for RPC operations. The user is given permission (or not) to invoke the RPC operation with any parameters. For example, if each client is only allowed to reset their own interface, then NACM cannot be used.

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

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

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

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

4.27. Updating YANG Modules (Published versus Unpublished)

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

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

5. IANA-Maintained Modules

5.1. Context

IANA maintains a set of registries that are key for interoperability. The content of these registries are usually available using various formats (e.g., plain text, XML). However, there were some confusion in the past about whether the content of some registries is dependent on a specific representation format. For example, Section 5 of [RFC8892] was published to clarify that MIB and YANG modules are merely additional formats in which the "Interface Types (ifType)" and "Tunnel Types (tunnelType)" registries are available. The MIB [RFC2863] and YANG modules [RFC7224][RFC8675] are not separate registries, and the same values are always present in all formats of the same registry.

Also, some YANG modules include parameters and values directly in a module that is not maintained by IANA while these are populated in an IANA registry. Such a design is suboptimal as it creates another source of information that may deviate from the IANA registry as new values are assigned or some values are deprecated.

For the sake of consistency, better flexibility to support new values, and maintaining IANA registries as the unique authoritative source of information, when such an information is maintained in a registry, this document encourages the use of IANA-maintained modules.

The following section provides a set of guidelines for YANG module authors related to the design of IANA-maintained modules. These guidelines are meant to leverage existing IANA registries and use YANG as another format to present the content of these registries when appropriate.

5.2. Guidelines for IANA-Maintained Modules

When designing a YANG module for a functionality governed by a protocol for which IANA maintains a registry, it is RECOMMENDED to specify an IANA-maintained module that echoes the content of that registry. This is superior to including that content in an IETF-maintained module.

When one or multiple sub-registries are available under the same registry, it is RECOMMENDED to define an IANA-maintained module for each sub-registry. However, module designers MAY consider defining one single IANA-maintained module that covers all sub-registries if maintaining that single module is manageable (e.g., very few values are present or expected to be present for each sub-registry). An example of such a module is documented in Section 5.2 of [RFC9132].

An IANA-maintained module may use identities (e.g., [RFC8675]) or enumerations (e.g., [RFC9108]). The decision about which type to use is left to the module designers and should be made based upon specifics related to the intended use of the IANA-maintained module. For example, identities are useful if the registry entries are organized hierarchically, possibly including multiple inheritances. It is RECOMMENDED that the reasoning for the design choice is documented in the companion specification that registers an IANA-maintained module. For example, [RFC9244] defines an IANA-maintained module that uses enumerations for the following reason:

> "The DOTS telemetry module (Section 10.1) uses "enumerations" rather than "identities" to define units, samples, and intervals because otherwise the namespace identifier "ietf-dots-telemetry" must be included when a telemetry attribute is included (e.g., in a mitigation efficacy update). The use of "identities" is thus suboptimal from a message compactness standpoint; one of the key requirements for DOTS messages."

Designers of IANA-maintained modules MAY supply the full initial version of the module in a specification document that registers the module or only a script to be used (including by IANA) for generating the module (e.g., an XSLT stylesheet as in Appendix A of [RFC9108]). For both cases, the document that defines an IANA-maintained module MUST include a note indicating that the document is only documenting the initial version of the module and that the authoritative version is to be retrieved from the IANA registry. It is RECOMMENDED to include the URL from where to retrieve the recent version of the module. When a script is used, the Internet-Draft that defines an IANA-maintained module SHOULD include an appendix with the initial full version of the module. Including such an appendix in pre-RFC versions is meant to assess the correctness of the outcome of the supplied script. The authors MUST include a note to the RFC Editor requesting that the appendix be removed before publication as RFC. Initial versions of IANA-maintained modules that are published in RFCs may be misused despite the appropriate language to refer to the IANA registry to retrieve the up-to-date module. This is problematic for interoperability, e.g., when values are deprecated or are associated with a new meaning.

> Note: [Style] provides XSLT 1.0 stylesheets and other tools for translating IANA registries to YANG modules. The tools can be used to generate up-to-date revisions of an IANA-maintained module based upon the XML representation of an IANA registry.

If an IANA-maintained module is imported by another module, a normative reference with the IANA URL from where to retrieve the IANA-maintained module SHOULD be included. Although not encouraged, referencing the RFC that defines the initial version of the IANA module is acceptable in specific cases (e.g., the imported version is specifically the initial version, the RFC includes useful description about the usage of the module).

Examples of IANA URLs from where to retrieve the latest version of an IANA-maintained module are: [IANA_BGP-L2_URL], [IANA_PW-Types_URL], and [IANA_BFD_URL]. [IANA_FOO_URL] is used in the following to refer to such URLs. These URLs are expected to be sufficiently permanent and stable.

5.3. Guidance for Writing the IANA Considerations for RFCs Defining IANA-Maintained Modules

In addition to the IANA considerations in Section 3.8, the IANA Considerations Section of an RFC that includes an IANA- maintained module MUST provide the required instructions for IANA to automatically perform the maintenance of that IANA module. These instructions describe how to proceed with updates to the IANA-maintained module that are triggered by a change to the authoritative registry. Concretely, the IANA Considerations Section SHALL at least provide the following information:

* An IANA request to add a note to the page displaying the information about the IANA-maintained module that new values must not be directly added to the module, but to an authoritative IANA registry.

* An IANA request to add a note to the authoritative IANA registry to indicate that any change to the registry must be reflected into the corresponding IANA-maintained module.

* Details about the required actions (e.g., add a new "identity" or "enum" statement) to update the IANA-maintained module to reflect changes to an authoritative IANA registry. Typically, these details have to include the procedure to create a new "identity" statement name and sub-statements ("base", "status", "description", and "reference") or a new "enum" statement and sub-statements ("value", "status", "description", and "reference").

* A note that unassigned or reserved values must not be present in the IANA-maintained module.

* An indication whether experimental values are included in the IANA-maintained module. Absent such an indication, experimental values MUST NOT be listed in the IANA-maintained module.

* An instruction about how to generate the "revision" statement.

A template for the IANA Considerations is provided in Section 5.3.1 for IANA-maintained modules with identities and Section 5.3.2 for IANA- maintained modules with enumerations. Authors may modify the template to reflect specifics of their modules (e.g., Multiple registries can be listed for a single IANA-maintained module, no explicit description (or name) field is listed under the authoritative IANA registry).

The following templates are to be considered in addition to the required information that is provided in Section 3.8.

5.3.1. Template for IANA-Maintained Modules with Identities

```
<CODE BEGINS>
This document defines the initial version of the IANA-maintained
"iana-foo" YANG module.  The most recent version of the YANG module
is available from the "YANG Parameters" registry
[IANA-YANG-PARAMETERS].

IANA is requested to add this note to the registry:

   New values must not be directly added to the "iana-foo" YANG
   module.  They must instead be added to the "foo" registry.

When a value is added to the "foo" registry, a new "identity"
statement must be added to the "iana-foo" YANG module.  The name of
the "identity" is the lower-case of the name provided in the
registry.  The "identity" statement should have the following sub-
statements defined:

 "base":        Contains 'name-base-identity-defined-in-foo'.

 "status":      Include only if a registration has been deprecated or
                obsoleted.  IANA "deprecated" maps to YANG status
                "deprecated", and IANA "obsolete" maps to YANG status
                "obsolete".

 "description":  Replicates the description from the registry.

 "reference":   Replicates the reference(s) from the registry with the
                title of the document(s) added.

Unassigned or reserved values are not present in the module.

When the "iana-foo" YANG module is updated, a new "revision"
statement with a unique revision date must be added in front of the
existing revision statements.

IANA is requested to add this note to [reference-to-the-iana-foo-
registry]:

   When this registry is modified, the YANG module "iana-foo"
   [IANA_FOO_URL] must be updated as defined in RFCXXXX.
<CODE ENDS>
```

5.3.2. Template for IANA-Maintained Modules with Enumerations

```
<CODE BEGINS>
This document defines the initial version of the IANA-maintained
"iana-foo" YANG module.  The most recent version of the YANG module
is available from the "YANG Parameters" registry
[IANA-YANG-PARAMETERS].

IANA is requested to add this note to the registry:

    New values must not be directly added to the "iana-foo" YANG
    module.  They must instead be added to the "foo" registry.

When a value is added to the "foo" registry, a new "enum" statement
must be added to the "iana-foo" YANG module.  The "enum" statement,
and sub-statements thereof, should be defined:

 "enum":        Replicates a name from the registry.

 "value":       Contains the decimal value of the IANA-assigned value.

 "status":      Is included only if a registration has been deprecated
                or obsoleted.  IANA "deprecated" maps to YANG status
                "deprecated", and IANA "obsolete" maps to YANG status
                "obsolete".

 "description":  Replicates the description from the registry.

 "reference":   Replicates the reference(s) from the registry with the
                title of the document(s) added.

Unassigned or reserved values are not present in the module.

When the "iana-foo" YANG module is updated, a new "revision"
statement with a unique revision date must be added in front of the
existing revision statements.

IANA is requested to add this note to [reference-to-the-iana-foo-
registry]:

    When this registry is modified, the YANG module "iana-foo"
    [IANA_FOO_URL] must be updated as defined in RFCXXXX.
<CODE ENDS>
```

6. IANA Considerations

The following registration in the "ns" subregistry of the "IETF XML Registry" [RFC3688] was detailed in [RFC6087]. This document requests IANA to update this registration to reference this document.

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

The following assignment was detailed in [RFC6087] and has been updated by IANA in the "YANG Module Names" registry to reference [RFC8407]. This document requests IANA to update the reference for the "YANG Module Names" registry to point to the RFC number that will be assigned to this document as it contains the template necessary for registration in Appendix B.

Field	Value
Name	ietf-template
Namespace	urn:ietf:params:xml:ns:yang:ietf-template
Prefix	temp
Maintained by IANA?	N
Reference	RFC XXXX

Table 2: YANG Registry Assignment

7. Security Considerations

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

8. References

8.1. Normative References

[ID-Guidelines] IETF, "Guidelines to Authors of Internet-Drafts", n.d., <https://authors.ietf.org/en/content-guidelines-overview>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

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

[RFC5378] Bradner, S., Ed. and J. Contreras, Ed., "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, DOI 10.17487/RFC5378, November 2008, <https://www.rfc-editor.org/rfc/rfc5378>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/rfc/rfc6242>.

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

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

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

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

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

[RFC8791] Bierman, A., Björklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/rfc/rfc8791>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

[W3C.REC-xpath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

8.2. Informative References

[IANA-MOD-NAMES] IANA, "YANG Module Names", <https://www.iana.org/assignments/yang-parameters/>.

[IANA-XML] IANA, "IETF XML Registry", <https://www.iana.org/assignments/xml-registry/>.

[IANA-YANG-PARAMETERS] "YANG Parameters", n.d., <https://www.iana.org/assignments/yang-parameters>.

[IANA_BFD_URL] IANA, "iana-bfd-types YANG Module", <https://www.iana.org/assignments/iana-bfd-types/iana-bfd-types.xhtml>.

[IANA_BGP-L2_URL]
IANA, "iana-bgp-l2-encaps YANG Module",
<https://www.iana.org/assignments/iana-bgp-l2-encaps/iana-bgp-l2-encaps.xhtml>.

[IANA_PW-Types_URL]
IANA, "iana-pseudowire-types YANG Module",
<https://www.iana.org/assignments/iana-pseudowire-types/iana-pseudowire-types.xhtml>.

[RFC-STYLE]
RFC Editor, "Style Guide",
<http://www.rfc-editor.org/styleguide/>.

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

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <https://www.rfc-editor.org/rfc/rfc2863>.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/rfc/rfc3849>.

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

[RFC4181] Heard, C., Ed., "Guidelines for Authors and Reviewers of MIB Documents", BCP 111, RFC 4181, DOI 10.17487/RFC4181, September 2005, <https://www.rfc-editor.org/rfc/rfc4181>.

[RFC5737] Arkko, J., Cotton, M., and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation", RFC 5737, DOI 10.17487/RFC5737, January 2010, <https://www.rfc-editor.org/rfc/rfc5737>.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <https://www.rfc-editor.org/rfc/rfc6087>.

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

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/rfc/rfc7223>.

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

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

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

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/rfc/rfc7951>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/rfc/rfc8340>.

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

[RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <https://www.rfc-editor.org/rfc/rfc8349>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/rfc/rfc8407>.

[RFC8675] Boucadair, M., Farrer, I., and R. Asati, "A YANG Data Model for Tunnel Interface Types", RFC 8675, DOI 10.17487/RFC8675, November 2019, <https://www.rfc-editor.org/rfc/rfc8675>.

[RFC8892] Thaler, D. and D. Romascanu, "Guidelines and Registration Procedures for Interface Types and Tunnel Types", RFC 8892, DOI 10.17487/RFC8892, August 2020, <https://www.rfc-editor.org/rfc/rfc8892>.

[RFC9108] Lhotka, L. and P. paek, "YANG Types for DNS Classes and Resource Record Types", RFC 9108, DOI 10.17487/RFC9108, September 2021, <https://www.rfc-editor.org/rfc/rfc9108>.

[RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <https://www.rfc-editor.org/rfc/rfc9132>.

[RFC9244] Boucadair, M., Ed., Reddy.K, T., Ed., Doron, E., Chen, M., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", RFC 9244, DOI 10.17487/RFC9244, June 2022, <https://www.rfc-editor.org/rfc/rfc9244>.

[RFC9291] Boucadair, M., Ed., Gonzalez de Dios, O., Ed., Barguil, S., and L. Munoz, "A YANG Network Data Model for Layer 2 VPNs", RFC 9291, DOI 10.17487/RFC9291, September 2022, <https://www.rfc-editor.org/rfc/rfc9291>.

[Style] "IANA YANG", n.d., <https://github.com/llhotka/iana-yang>.

Appendix A. Module Review Checklist

This section is adapted from [RFC4181].

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

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

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

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

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

* Security Considerations section -- verify that the document uses the latest approved template from the Operations and Management (OPS) area website (see <https://trac.ietf.org/area/ops/trac/wiki/yang-security-guidelines>) and that the guidelines therein have been followed.

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

  XML Namespace Registry: Register the YANG module namespace.

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

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

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

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

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

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

  <https://trac.ietf.org/trac/netconf/wiki>

  and

  <https://trac.ietf.org/trac/netmod/wiki>

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

Appendix B. YANG Module Template

```
<CODE BEGINS> file "ietf-template@2016-03-20.yang"
module ietf-template {
  yang-version 1.1;

  // replace this string with a unique namespace URN value

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

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

  prefix temp;

  // import statements here: e.g.,
  // import ietf-yang-types { prefix yang; }
  // import ietf-inet-types { prefix inet; }
  // identify the IETF working group if applicable

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

  // update this contact statement with your info

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/your-wg-name/>
     WG List:  <mailto:your-wg-name@ietf.org>

     Editor:   your-name
```

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

    // replace the first sentence in this description statement.
    // replace the copyright notice with the most recent
    // version, if it has been updated since the publication
    // of this document

    description
      "This module defines a template for other YANG modules.

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

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the 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; see
       the RFC itself for full legal notices.";

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

    // replace '2016-03-20' with the module publication date
    // the format is (year-month-day)

    revision 2016-03-20 {
      description
        "what changed in this revision";
      reference "RFC XXXX: <Replace With Document Title>";
    }

    // extension statements
    // feature statements
    // identity statements
    // typedef statements
    // grouping statements
    // data definition statements
    // augment statements
    // rpc statements
    // notification statements
    // DO NOT put deviation statements in a published module
  }
  <CODE ENDS>
```

Acknowledgments

Thanks to Jürgen Schönwälder, Ladislav Lhotka, and Qin Wu for the discussion and valuable comments. Special thanks to Ladislav Lhotka for sharing more context that led to the design documented in [RFC9108].

Thanks to Andy Bierman, Italo Busi, Benoit Claise, Tom Petch, and Randy Presuhn for the comments. Lou Berger suggested to include more details about IANA considerations.

The author of RFC 8407: Andy Bierman

YumaWorks

email: andy@yumaworks.com

Acknowledgments from RFC 8407: The structure and contents of this document are adapted from "Guidelines for Authors and Reviewers of MIB Documents" [RFC4181], by C. M. Heard.

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

Authors' Addresses

Mohamed Boucadair
Orange
France
Email: mohamed.boucadair@orange.com

Qin Wu
Huawei
China
Email: bill.wu@huawei.com

netmod
Internet-Draft
Intended status: Standards Track
Expires: 29 December 2023

O. G. D. Dios
S. Barguil
Telefonica
M. Boucadair
Orange
Q. Wu
Huawei

27 June 2023

Extensions to the Access Control Lists (ACLs) YANG Model
draft-ietf-netmod-acl-extensions-02

Abstract

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

The document also defines an IANA-maintained module for ICMP types.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Network Modeling Working Group mailing list (netmod@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netmod/.

Source for this draft and an issue tracker can be found at https://github.com/boucadair/enhanced-acl-netmod.

Status of This Memo

Table of Contents

1. Introduction

[RFC8519] defines Access Control Lists (ACLs) as a user-ordered set of filtering rules. The model targets the configuration of the filtering behavior of a device. However, the model structure, as defined in [RFC8519], suffers from a set of limitations. This document describes these limitations and proposes an enhanced ACL structure. The YANG module in this document is solely based on augmentations to the ACL YANG module defined in [RFC8519].

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

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

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

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

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

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

The document also defines an IANA-maintained module for ICMP types. The design of the module adheres with the recommendations in [I-D.boucadair-netmod-iana-registries]. A template to generate the module is available at Appendix A. Readers should refer to the IANA website [REF_TBC] to retrieve the latest version of the module. The module is provided in Appendix B for the users convenience, but that appendix will be removed from the final RFC.

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

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

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

3. Problem Statement & Gap Analysis

3.1. Suboptimal Configuration: Lack of Support for Lists of Prefixes

IP prefix-related data nodes, e.g., "destination-ipv4-network" or "destination-ipv6-network", do not support handling a list of IP prefixes, which may then lead to having to support large numbers of ACL entries in a configuration file.

The same issue is encountered when ACLs have to be in place to mitigate DDoS attacks that involve a set of sources (e.g., [RFC9132]). The situation is even worse when both a list of sources and destination prefixes are involved in the filtering.

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

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

```
                  "ipv6": {
                    "destination-ipv6-network":
                      "2001:db8:6401:c::/64",
                    "source-ipv6-network":
                      "2001:db8:1234::/96",
                    "protocol": 17,
                    "flow-label": 10000
                  },
                  "udp": {
                    "source-port": {
                      "operator": "lte",
                      "port": 80
                    },
                    "destination-port": {
                      "operator": "neq",
                      "port": 1010
                    }
                  }
                },
                "actions": {
                  "forwarding": "accept"
                }
              }
            ]
          }
        }
      ]
    }
  }
```

Figure 1: Example Illustrating Sub-optimal Use of the ACL Model with a Prefix List (Message Body)

Such a configuration is suboptimal for both:

* Network controllers that need to manipulate large files. All or a subset for this configuration will need to be passed to the underlying network devices.

* Devices may receive such a configuration and thus will need to maintain it locally.

Figure 2 depicts an example of an optimized structure:

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

Figure 2: Example Illustrating Optimal Use of the ACL Model in a Network Context (Message Body)

3.2. Manageability: Impossibility to Use Aliases or Defined Sets

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

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

* Prefix sets: Used to create lists of IPv4 or IPv6 prefixes.

* Protocol sets: Used to create a list of protocols.

* Port number sets: Used to create lists of TCP or UDP port values (or any other transport protocol that makes uses of port numbers). The identity of the protocols is identified by the protocol set, if present. Otherwise, a set applies to any protocol.

* ICMP sets: Uses to create lists of ICMP-based filters. This applies only when the protocol is set to ICMP or ICMPv6.

A candidate structure is shown in Figure 3:

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

Figure 3: Examples of Defined Sets

Aliases may also be considered to manage resources that are identified by a combination of various parameters as shown in the candidate tree in Figure 4. Note that some aliases can be provided by decomposing them into separate sets.

```
|  +--rw aliases
|  |  +--rw alias* [name]
|  |     +--rw name                  string
|  |     +--rw prefix*       inet:ip-prefix
|  |     +--rw port-range* [lower-port]
|  |     |  +--rw lower-port    inet:port-number
|  |     |  +--rw upper-port?   inet:port-number
|  |     +--rw protocol*     uint8
|  |     +--rw fqdn*         inet:domain-name
|  |     +--rw uri*          inet:uri
```

Figure 4: Examples of Aliases

3.3. Bind ACLs to Devices, Not Only Interfaces

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

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

3.4. Partial or Lack of IPv4/IPv6 Fragment Handling

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

Defining a new IPv4/IPv6 matching field called 'fragment' is thus required to efficiently handle fragment-related filtering rules.

3.5. Suboptimal TCP Flags Handling

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

3.6. Rate-Limit Action

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

3.7. Payload-based Filtering

Some transport protocols use existing protocols (e.g., TCP or UDP) as substrate. The match criteria for such protocols may rely upon the 'protocol' under '13', TCP/UDP match criteria, part of the TCP/UDP payload, or a combination thereof. [RFC8519] does not support matching based on the payload.

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

3.8. Reuse the ACLs Content Across Several Devices

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

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

* An ACL name can be used at both network and device levels.

* An ACL content updated at the network level should imply a transaction that updates the relevant content in all the nodes using this ACL.

* ACLs defined at the device level have a local meaning for the specific node.

* A device can be associated with a router, a VRF, a logical system, or a virtual node. ACLs can be applied in physical and logical infrastructure.

3.9. Match MPLS Headers

The ACLs could be used to create rules to match MPLS fields on a packet.

4. Overall Module Structure

4.1. Enhanced ACL

Figure 5 shows the full enhanced ACL tree:

```
module: ietf-acl-enh
  +--rw defined-sets
  |  +--rw ipv4-prefix-sets
  |  |  +--rw prefix-set* [name]
  |  |     +--rw name           string
  |  |     +--rw description?   string
  |  |     +--rw prefix*        inet:ipv4-prefix
  |  +--rw ipv6-prefix-sets
  |  |  +--rw prefix-set* [name]
  |  |     +--rw name           string
  |  |     +--rw description?   string
  |  |     +--rw prefix*        inet:ipv6-prefix
  |  +--rw port-sets
  |  |  +--rw port-set* [name]
  |  |     +--rw name    string
  |  |     +--rw port* [id]
  |  |        +--rw id                                string
  |  |        +--rw (port)?
  |  |           +--:(port-range-or-operator)
  |  |              +--rw port-range-or-operator
  |  |                 +--rw (port-range-or-operator)?
  |  |                    +--:(range)
  |  |                    |  +--rw lower-port    inet:port-number
  |  |                    |  +--rw upper-port    inet:port-number
  |  |                    +--:(operator)
  |  |                       +--rw operator?     operator
  |  |                       +--rw port          inet:port-number
  |  +--rw protocol-sets
  |  |  +--rw protocol-set* [name]
  |  |     +--rw name        string
  |  |     +--rw protocol*   union
  |  +--rw icmp-type-sets
  |     +--rw icmp-type-set* [name]
  |        +--rw name     string
  |        +--rw types* [type]
  |           +--rw type              uint8
  |           +--rw code?             uint8
  |
```

```
     |           +--rw rest-of-header?   binary
     +--rw aliases
        +--rw alias* [name]
           +--rw name          string
           +--rw prefix*       inet:ip-prefix
           +--rw port-range* [lower-port]
           |  +--rw lower-port    inet:port-number
           |  +--rw upper-port?   inet:port-number
           +--rw protocol*     uint8
           +--rw fqdn*         inet:domain-name
           +--rw uri*          inet:uri

     augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
       +--rw (payload)?
       |  +--:(prefix-pattern)
       |     +--rw prefix-pattern {match-on-payload}?
       |        +--rw offset?       identityref
       |        +--rw offset-end?   uint64
       |        +--rw operator?     operator
       |        +--rw prefix?       binary
       +--rw (alias)?
       |  +--rw alias-name*        alias-ref
       +--rw (mpls)?
          +--:(mpls-values)
             +--rw mpls-values {match-on-mpls}?
                +--rw traffic-class?       uint8
                +--rw label-position       identityref
                +--rw upper-label-range?   uint32
                +--rw lower-label-range?   uint32
                +--rw label-block-name     string
                +--rw ttl-value?           uint8
     augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l2:
       +--rw vlan-filter {match-on-vlan-filter}?
          +--rw frame-type?         string
          +--rw (vlan-type)?
             +--:(range)
             |  +--rw lower-vlan    uint16
             |  +--rw upper-vlan    uint16
             +--:(operator)
                +--rw operator?     packet-fields:operator
                +--rw vlan*         uint16
     augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l2:
       +--rw isid-filter {match-on-isid-filter}?
          +--rw (isid-type)?
             +--:(range)
             |  +--rw lower-isid    uint16
             |  +--rw upper-isid    uint16
             +--:(operator)
```

```
            +--rw operator?     packet-fields:operator
            +--rw isid*         uint16
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l3
              /acl:ipv4:
      +--rw ipv4-fragment
      |  +--rw operator?   operator
      |  +--rw type?       fragment-type
      +--rw source-ipv4-prefix-list?        ipv4-prefix-set-ref
      +--rw destination-ipv4-prefix-list?   ipv4-prefix-set-ref
      +--rw next-header-set?                protocol-set-ref
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l3
              /acl:ipv6:
      +--rw ipv6-fragment
      |  +--rw operator?   operator
      |  +--rw type?       fragment-type
      +--rw source-ipv6-prefix-list?        ipv6-prefix-set-ref
      +--rw destination-ipv6-prefix-list?   ipv6-prefix-set-ref
      +--rw protocol-set?                   protocol-set-ref
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l4
              /acl:tcp:
      +--rw flags-bitmask
      |  +--rw (mode)?
      |     +--:(explicit)
      |     |  +--rw operator?            operator
      |     |  +--rw explicit-tcp-flag*   identityref
      |     +--:(builtin)
      |        +--rw bitmask?             uint16
      +--rw source-tcp-port-set?        port-set-ref
      +--rw destination-tcp-port-set?   port-set-ref
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l4
              /acl:udp:
      +--rw source-udp-port-set?        port-set-ref
      +--rw destination-udp-port-set?   port-set-ref
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches/acl:l4
              /acl:icmp:
      +--rw icmp-set?   icmp-type-set-ref
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:actions:
      +--rw rate-limit?   decimal64
```

Figure 5: Enhanced ACL tree

4.2. Defined sets

The augmented ACL structure includes several containers to manage reusable sets of elements that can be matched in an ACL entry. Each set is uniquely identified by a name, and can be called from the relevant entry. The following sets are defined:

* IPv4 prefix set: It contains a list of IPv4 prefixes. A match will be considered if the IP address (source or destination, depending on the ACL entry) is contained in any of the prefixes.

* IPv6 prefix set: It contains a list of IPv6 prefixes. A match will be considered if the IP address (source or destination, depending on the ACL entry) is contained in any of the prefixes.

* Port sets: It contains a list of port numbers to be used in TCP / UDP entries. The ports can be individual port numbers, a range of ports, and an operation.

* Protocol sets: It contains a list of protocol values. Each protocol can be identified either by a number (e.g., 17) or a name (e.g., UDP).

* ICMP sets: It contains a list of ICMP types, each of them identified by a type value, optionally the code and the rest of the header.

4.3. TCP Flags Handling

The augmented ACL structure includes a new leaf 'flags-bitmask' to better handle flags.

Clients that support both 'flags-bitmask' and 'flags' matching fields MUST NOT set these fields in the same request.

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

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

Figure 6: Example to Deny TCP Null Attack Messages (Request Body)

4.4. Fragments Handling

The augmented ACL structure includes a new leaf 'fragment' to better handle fragments.

Clients that support both 'fragment' and 'flags' matching fields MUST NOT set these fields in the same request.

Figure 7 shows the content of a POST request to allow the traffic destined to 198.51.100.0/24 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

* "drop-all-fragments" ACE: discards all fragments.

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

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

Figure 7: Example Illustrating Candidate Filtering of IPv4 Fragmented Packets (Message Body)

Figure 8 shows an example of the body of a POST request to allow the traffic destined to 2001:db8::/32 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

* "drop-all-fragments" ACE: discards all fragments (including atomic fragments). That is, IPv6 packets that include a Fragment header (44) are dropped.

* "allow-dns-packets" ACE: accepts DNS packets destined to 2001:db8::/32.

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

Figure 8: Example Illustrating Candidate Filtering of IPv6 Fragmented Packets (Message Body)

4.5. Rate-Limit Traffic

In order to support rate-limiting (see Section 3.6), a new action called "rate-limit" is defined. Figure 9 shows an ACL example to rate-limit incoming SYNs during a SYN flood attack.

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

Figure 9: Example Rate-Limit Incoming TCP SYNs (Message Body).

4.6. ISID Filter

Provider backbone bridging (PBB) was originally defined as Virtual Bridged Local Area Networks [IEEE802.1ah] standard. However, instead of multiplexing VLANs, PBB duplicates the MAC layer of the customer frame and separates it from the provider domain, by encapsulating it in a 24 bit instance service identifier (I-SID). This provides for more transparency between the customer network and the provider network.

The I-component forms the customer or access facing interface or routing instance. The I-component is responsible for mapping customer Ethernet traffic to the appropriate I-SID. In the network is mandatory to configure the default service identifier.

Being able to filter by I-component Service identifier is a feature of the EVNP-PBB configuration.

Figure 10 shows an ACL example to illustrate the ISID range filtering.

```
{
  "ietf-acces-control-list:acls": {
        "acl": [
          {
            "name": "test",
            "aces": {
              "ace": [
                {
                  "name": "1",
                  "matches": {
                    "ietf-acl-enh:isid-filter": {
                      "lower-isid": 100,
                      "upper-isid": 200
                    }
                  },
                  "actions": {
                    "forwarding": "ietf-acces-control-list:accept"
                  }
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```

Figure 10: Example ISID Filter (Message Body)

4.7. VLAN Filter

Being able to filter all packets that are bridged within a VLAN or that are routed into or out of a bridge domain is part of the VPN control requirements derived of the EVPN definition done in [RFC7209]. So, all packets that are bridged within a VLAN or that are routed into or out of a VLAN can be captured, forwarded, translated or discarded based on the network policy applied.

Figure 11 shows an ACL example to illustrate how to apply a VLAN range filter.

```
{
  "ietf-acces-control-list:acls": {
    "acl": [
      {
        "name": "VLAN_FILTER",
        "aces": {
          "ace": [
            {
              "name": "1",
              "matches": {
                "ietf-acl-enh:vlan-filter": {
                  "lower-vlan": 10,
                  "upper-vlan": 20
                }
              },
              "actions": {
                "forwarding": "ietf-acces-control-list:accept"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Figure 11: Example of VLAN Filter (Message Body)

4.8. Match MPLS Headers

The ACL models can be used to create rules to match MPLS fields on a packet. The MPLS headers defined in [RFC3032] and [RFC5462] contains the following fields:

* Traffic Class: 3 bits ’EXP’ renamed to ’Traffic Class Field."

* Label Value: A 20-bit field that carries the actual value of the MPLS Label.

* TTL: An eight-bit field that is used to encode a time-to-live value.

The structure of the MPLS ACL subtree is shown in Figure 12:

```
augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
  ...
  +--rw (mpls)?
     +--:(mpls-values)
        +--rw mpls-values {match-on-mpls}?
           +--rw traffic-class?       uint8
           +--rw label-position       identityref
           +--rw upper-label-range?   uint32
           +--rw lower-label-range?   uint32
           +--rw label-block-name     string
           +--rw ttl-value?           uint8
```

Figure 12: MPLS Header Match Subtree

5. YANG Modules

5.1. Enhanced ACL

This model imports types from [RFC6991], [RFC8519], and [RFC8294].

<CODE BEGINS>
file ietf-acl-enh@2022-10-24.yang

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

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-access-control-list {
    prefix acl;
    reference
      "RFC 8519: YANG Data Model for Network Access
                 Control Lists (ACLs), Section 4.1";
  }
  import ietf-packet-fields {
    prefix packet-fields;
    reference
      "RFC 8519: YANG Data Model for Network Access
                 Control Lists (ACLs), Section 4.2";
  }

  import ietf-routing-types {
    prefix rt-types;
```

```
      reference
        "RFC 8294: Common YANG Data Types for the Routing Area";
      }

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

       Author:    Mohamed Boucadair
                  mailto:mohamed.boucadair@orange.com
       Author:    Samier Barguil
                  mailto:samier.barguilgiraldo.ext@telefonica.com
       Author:    Oscar Gonzalez de Dios
                  mailto:oscar.gonzalezdedios@telefonica.com";
    description
      "This module contains YANG definitions for enhanced ACLs.

       Copyright (c) 2023 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
       (http://trustee.ietf.org/license-info).

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

    revision 2022-10-24 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: Extensions to the Access Control Lists (ACLs)
                   YANG Model";
    }

    feature match-on-payload {
      description
        "Match based on a pattern is supported.";
    }

    feature match-on-vlan-filter {
      description
        "Match based on a VLAN range of vlan list is supported.";
```

```
  }

  feature match-on-isid-filter {
    description
      "Match based on a ISID range of vlan list is supported.";
  }

  feature match-on-alias {
    description
      "Match based on aliases.";
  }

  feature match-on-mpls {
    description
      "Match based on MPLS headers.";
  }

  identity offset-type {
    description
      "Base identity for payload offset type.";
  }

  identity layer3 {
    base offset-type;
    description
      "The offset starts at the beginning of the IP header.";
  }

  identity layer4 {
    base offset-type;
    description
      "The offset start right after the IP header. This can be
       typically the beginning of transport header (e.g., TCP
       or UDP).";
  }

  identity payload {
    base offset-type;
    description
      "The offset start right after the end of the transport
       payload. For example, this represents the beginning of the
       TCP data right after any TCP options or the beginning of
       the UDP payload right after the UDP header.";
  }

  identity tcp-flag {
    description
      "Base Identity for the TCP Flags.";
```

```
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity ack {
    base tcp-flag;
    description
      "Acknowledgment TCP flag bit.";
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity syn {
    base tcp-flag;
    description
      "Synchronize sequence numbers.";
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity fin {
    base tcp-flag;
    description
      "No more data from the sender.";
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity urg {
    base tcp-flag;
    description
      "Urgent pointer TCP flag bit.";
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity psh {
    base tcp-flag;
    description
      "The Push function flag is similar to the URG flag and tells
       the receiver to process these packets as they are received
       instead of buffering them.";
    reference
      "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
  }

  identity rst {
    base tcp-flag;
```

```
      description
        "Reset TCP flag bit.";
      reference
        "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
    }

    identity ece {
      base tcp-flag;
      description
        "ECN-Echo TCP flag bit.";
      reference
        "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
    }

    identity cwr {
      base tcp-flag;
      description
        "Congestion Window Reduced flag bit";
      reference
        "RFC 9293: Transmission Control Protocol (TCP), Section 3.1";
    }

    identity mpls-acl-type {
      base acl:acl-base;
      description
        "An ACL that matches on fields from the MPLS header.";
    }

    identity label-position {
      description
        "Base identity for deriving MPLS label position.";
    }

    identity top {
      base label-position;
      description
        "Top of the label stack.";
    }

    identity bottom {
      base label-position;
      description
        "Bottom of the label stack.";
    }

    typedef operator {
      type bits {
        bit not {
```

```
          position 0;
          description
            "If set, logical negation of operation.";
        }
        bit match {
          position 1;
          description
            "Match bit.  This is a bitwise match operation defined as
             '(data & value) == value'.";
        }
        bit any {
          position 2;
          description
            "Any bit.  This is a match on any of the bits in  bitmask.
             It evaluates to 'true' if any of the bits in the value mask
             are set in the data,  i.e., '(data & value) != 0'.";
        }
      }
      description
        "Specifies how to apply the defined bitmask.
         'any' and 'match' bits must not be set simultaneously.";
    }

    typedef fragment-type {
      type bits {
        bit df {
          position 0;
          description
            "Don't fragment bit for IPv4.
             Must be set to 0 when it appears in an IPv6 filter.";
        }
        bit isf {
          position 1;
          description
            "Is a fragment.";
        }
        bit ff {
          position 2;
          description
            "First fragment.";
        }
        bit lf {
          position 3;
          description
            "Last fragment.";
        }
      }
      description
```

```
        "Different fragment types to match against.";
    }

    typedef ipv4-prefix-set-ref {
      type leafref {
        path "/acl-enh:defined-sets/acl-enh:ipv4-prefix-sets"
           + "/acl-enh:prefix-set/acl-enh:name";
      }
      description
        "Defines a reference to an IPv4 prefix set.";
    }

    typedef ipv6-prefix-set-ref {
      type leafref {
        path "/acl-enh:defined-sets/acl-enh:ipv6-prefix-sets"
           + "/acl-enh:prefix-set/acl-enh:name";
      }
      description
        "Defines a reference to an IPv6 prefix set.";
    }

    typedef port-set-ref {
      type leafref {
        path "/acl-enh:defined-sets/acl-enh:port-sets"
           + "/acl-enh:port-set/acl-enh:name";
      }
      description
        "Defines a reference to a port set.";
    }

    typedef protocol-set-ref {
      type leafref {
        path "/acl-enh:defined-sets/acl-enh:protocol-sets"
           + "/acl-enh:protocol-set/acl-enh:name";
      }
      description
        "Defines a reference to a protocol set.";
    }

    typedef icmp-type-set-ref {
      type leafref {
        path "/acl-enh:defined-sets/acl-enh:icmp-type-sets"
           + "/acl-enh:icmp-type-set/acl-enh:name";
      }
      description
        "Defines a reference to an ICMP type set.";
    }
```

```
  typedef alias-ref {
    type leafref {
      path "/acl-enh:aliases/acl-enh:alias/acl-enh:name";
    }
    description
      "Defines a reference to an alias.";
  }

  grouping tcp-flags {
    description
      "Operations on TCP flags.";
    choice mode {
      description
        "Choice of how flags are indicated.";
      case explicit {
        leaf operator {
          type operator;
          default "match";
          description
            "How to interpret the TCP flags.";
        }
        leaf-list explicit-tcp-flag {
          type identityref {
            base tcp-flag;
          }
          description
            "An explicit list of the TCP flags that are to be
             matched.";
        }
      }
      case builtin {
        leaf bitmask {
          type uint16;
          description
            "The bitmask matches the last 4 bits of byte 12 and 13 of
             the TCP header.  For clarity, the 4 bits of byte 12
             corresponding to the TCP data offset field are not
             included in any matching.";
        }
      }
    }
  }

  grouping fragment-fields {
    description
      "Operations on fragment types.";
    leaf operator {
      type operator;
```

```
        default "match";
        description
          "How to interpret the fragment type.";
      }
      leaf type {
        type fragment-type;
        description
          "What fragment type to look for.";
      }
    }

  grouping mpls-match-parameters-config {
    description
      "Parameters for the configuration of MPLS match rules.";

    leaf traffic-class {
      type uint8 {
        range "0..7";
      }
      description
        "The value of the MPLS traffic class (TC) bits,
         formerly known as the EXP bits.";
    }

    leaf label-position {
      type identityref {
        base label-position;
      }
      description
        "Position of the label";
    }

    leaf upper-label-range {
      type rt-types:mpls-label;
      description
        "Match MPLS label value on the MPLS header.
         The usage of this field indicated the upper
         range value in the top of the stack.
         This label value does not include the
         encodings of Traffic Class and TTL.";
       reference
        "RFC 3032: MPLS Label Stack Encoding";
    }

    leaf lower-label-range {
      type rt-types:mpls-label;
      description
        "Match MPLS label value on the MPLS header.
```

```
          The usage of this field indicated the lower
          range value in the top of the stack.
          This label value does not include the
          encodings of Traffic Class and TTL.";
       reference
         "RFC 3032: MPLS Label Stack Encoding";
     }

     leaf label-block-name {
       type string;
       description
       "Reference to a label block predefiend in the
        implementation.";
     }

     leaf ttl-value {
       type uint8;
       description
         "Time-to-live MPLS packet value match.";
       reference
         "RFC 3032: MPLS Label Stack Encoding";
       }
     }

     grouping payload {
       description
         "Operations on payload match.";
       leaf offset {
         type identityref {
           base offset-type;
         }
         description
           "Indicates the payload offset. This will indicate the position
            of the data in packet to use for the match.";
       }
       leaf offset-end {
         type uint64;
         units "bytes";
         description
           "Indicates the number of bytes, starting from the offset to
            cover when performing the prefix match.";
       }
       leaf operator {
         type operator;
         default "match";
         description
           "How to interpret the prefix match.";
       }
```

```
      leaf prefix {
        type binary;
        description
          "The binary pattern to match against.";
      }
    }

    grouping alias {
      description
        "Specifies an alias.";
      leaf-list prefix {
        type inet:ip-prefix;
        description
          "IPv4 or IPv6 prefix of the alias.";
      }
      list port-range {
        key "lower-port";
        description
          "Port range.  When only lower-port is
           present, it represents a single port number.";
        leaf lower-port {
          type inet:port-number;
          mandatory true;
          description
            "Lower port number of the port range.";
        }
        leaf upper-port {
          type inet:port-number;
          must '. >= ../lower-port' {
            error-message
              "The upper-port number must be greater than
               or equal to the lower-port number.";
          }
          description
            "Upper port number of the port range.";
        }
      }
      leaf-list protocol {
        type uint8;
        description
          "Identifies the target protocol number.

           Values are taken from the IANA protocol registry:
           https://www.iana.org/assignments/protocol-numbers/

           For example, 6 for TCP or 17 for UDP.";
      }
      leaf-list fqdn {
```

```
        type inet:domain-name;
        description
          "FQDN identifying the target.";
      }
      leaf-list uri {
        type inet:uri;
        description
          "URI identifying the target.";
      }
    }

    augment "/acl:acls/acl:acl/acl:aces/acl:ace"
          + "/acl:matches" {
      description
        "Add a new match types.";
      choice payload {
        description
          "Match a prefix pattern.";
        container prefix-pattern {
          if-feature "match-on-payload";
          description
            "Rule to perform payload-based match.";
          uses payload;
        }
      }
      choice alias {
        description
          "Match on aliases.";
        leaf-list alias-name {
          type alias-ref;
          description
            "A set of aliases.";
        }
      }
      choice mpls {
        container mpls-values {
          if-feature "match-on-mpls";
          uses mpls-match-parameters-config;
          description
            "Rule set that matches MPLS headers.";
        }
        description
          "Match MPLS headers, for example, label values";
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:matches/acl:l2" {
```

```
      description
        "Handle the augmentation of MAC VLAN Filter.";
      container vlan-filter {
        if-feature "match-on-vlan-filter";
        description
          "Indicates how to handle MAC VLANs.";
        leaf frame-type {
          type string;
          description
            "Entering the frame type allows the
             filter to match a specific type of frame format";
        }
        choice vlan-type {
          description
            "vlan definition from range or operator.";
          case range {
            leaf lower-vlan {
              type uint16;
              must '. <= ../upper-vlan' {
                error-message
                  "The lower-vlan must be less than or equal to
                   the upper-vlan.";
              }
              mandatory true;
              description
                "Lower boundary for a vlan.";
            }
            leaf upper-vlan {
              type uint16;
              mandatory true;
              description
                "Upper boundary for a vlan.";
            }
          }
          case operator {
            leaf operator {
              type packet-fields:operator;
              default "eq";
              description
                "Operator to be applied on the vlan below.";
            }
            leaf-list vlan {
              type uint16;
              description
                "vlan number along with the operator on which to
                 match.";
            }
          }
```

```
        }
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:matches/acl:l2" {
      description
        "Handle the augmentation of ISID Filter.";
      container isid-filter {
        if-feature "match-on-isid-filter";
        description
          "Indicates how to handle ISID filters.
           The I-component is responsible for mapping customer
           Ethernet traffic to the appropriate ISID.";
        choice isid-type {
          description
            "ISID definition from range or operator.";
          case range {
            leaf lower-isid {
              type uint16;
              must '. <= ../upper-isid' {
                error-message
                  "The lower-vlan must be less than or equal to
                   the upper-isid.";
              }
              mandatory true;
              description
                "Lower boundary for a ISID.";
            }
            leaf upper-isid {
              type uint16;
              mandatory true;
              description
                "Upper boundary for a ISID.";
            }
          }
          case operator {
            leaf operator {
              type packet-fields:operator;
              default "eq";
              description
                "Operator to be applied on the ISID below.";
            }
            leaf-list isid {
              type uint16;
              description
                "ISID number along with the operator on which to
                 match.";
```

```
          }
        }
      }
    }
  }

  augment "/acl:acls/acl:acl/acl:aces"
        + "/acl:ace/acl:matches/acl:l3/acl:ipv4" {
    description
      "Handle non-initial and initial fragments for IPv4 packets.";
    container ipv4-fragment {
      description
        "Indicates how to handle IPv4 fragments.";
      uses fragment-fields;
    }
    leaf source-ipv4-prefix-list {
      type ipv4-prefix-set-ref;
      description
        "A reference to an IPv4 prefix list to match the source
         address.";
    }
    leaf destination-ipv4-prefix-list {
      type ipv4-prefix-set-ref;
      description
        "A reference to a prefix list to match the destination
         address.";
    }
    leaf next-header-set {
      type protocol-set-ref;
      description
        "A reference to a protocol set to match the next-header
         field.";
    }
  }

  augment "/acl:acls/acl:acl/acl:aces"
        + "/acl:ace/acl:matches/acl:l3/acl:ipv6" {
    description
      "Handles non-initial and initial fragments for IPv6 packets.";
    container ipv6-fragment {
      description
        "Indicates how to handle IPv6 fragments.";
      uses fragment-fields;
    }
    leaf source-ipv6-prefix-list {
      type ipv6-prefix-set-ref;
      description
        "A reference to a prefix list to match the source address.";
```

```
      }
      leaf destination-ipv6-prefix-list {
        type ipv6-prefix-set-ref;
        description
          "A reference to a prefix list to match the destination
           address.";
      }
      leaf protocol-set {
        type protocol-set-ref;
        description
          "A reference to a protocol set to match the protocol field.";
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:matches/acl:l4/acl:tcp" {
      description
        "Handles TCP flags and port sets.";
      container flags-bitmask {
        description
          "Indicates how to handle TCP flags.";
        uses tcp-flags;
      }
      leaf source-tcp-port-set {
        type port-set-ref;
        description
          "A reference to a port set to match the source port.";
      }
      leaf destination-tcp-port-set {
        type port-set-ref;
        description
          "A reference to a port set to match the destination port.";
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:matches/acl:l4/acl:udp" {
      description
        "Handle UDP port sets.";
      leaf source-udp-port-set {
        type port-set-ref;
        description
          "A reference to a port set to match the source port.";
      }
      leaf destination-udp-port-set {
        type port-set-ref;
        description
          "A reference to a port set to match the destination port.";
```

```
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:matches/acl:l4/acl:icmp" {
      description
        "Handle ICMP type sets.";
      leaf icmp-set {
        type icmp-type-set-ref;
        description
          "A reference to an ICMP type set to match the ICMP type
           field.";
      }
    }

    augment "/acl:acls/acl:acl/acl:aces"
          + "/acl:ace/acl:actions" {
      description
        "Rate-limit action.";
      leaf rate-limit {
        when "../acl:forwarding = 'acl:accept'" {
          description
            "Rate-limit valid only when accept action is used.";
        }
        type decimal64 {
          fraction-digits 2;
        }
        units "bytes per second";
        description
          "Indicates a rate-limit for the matched traffic.";
      }
    }


    container defined-sets {
      description
        "Predefined sets of attributes used in policy match
         statements.";
      container ipv4-prefix-sets {
        description
          "Data definitions for a list of IPv4 or IPv6
           prefixes which are matched as part of a policy.";
        list prefix-set {
          key "name";
          description
            "List of the defined prefix sets.";
          leaf name {
            type string;
```

```
            description
              "Name of the prefix set -- this is used as a label to
               reference the set in match conditions.";
          }
          leaf description {
            type string;
            description
              "Defined Set description.";
          }
          leaf-list prefix {
            type inet:ipv4-prefix;
            description
              "List of IPv4 prefixes to be used in match
               conditions.";
          }
        }
      }
      container ipv6-prefix-sets {
        description
          "Data definitions for a list of IPv6 prefixes which are
           matched as part of a policy.";
        list prefix-set {
          key "name";
          description
            "List of the defined prefix sets.";
          leaf name {
            type string;
            description
              "Name of the prefix set -- this is used as a label to
               reference the set in match conditions.";
          }
          leaf description {
            type string;
            description
              "A textual description of the prefix list.";
          }
          leaf-list prefix {
            type inet:ipv6-prefix;
            description
              "List of IPv6 prefixes to be used in match conditions.";
          }
        }
      }
      container port-sets {
        description
          "Data definitions for a list of ports which can
           be matched in policies.";
        list port-set {
```

```
          key "name";
          description
            "List of port set definitions.";
          leaf name {
            type string;
            description
              "Name of the port set -- this is used as a label to
               reference the set in match conditions.";
          }
          list port {
            key "id";
            description
              "Port numbers along with the operator on which to
               match.";
            leaf id {
              type string;
              description
                "Identifier of the list of port numbers.";
            }
            choice port {
              description
                "Choice of specifying the port number or referring to a
                 group of port numbers.";
              container port-range-or-operator {
                description
                  "Indicates a set of ports.";
                uses packet-fields:port-range-or-operator;
              }
            }
          }
        }
      }
      container protocol-sets {
        description
          "Data definitions for a list of protocols which can be matched
           in policies.";
        list protocol-set {
          key "name";
          description
            "List of protocol set definitions.";
          leaf name {
            type string;
            description
              "Name of the protocols set -- this is used as a label to
               reference the set in match conditions.";
          }
          leaf-list protocol {
            type union {
```

```
              type uint8;
              type string;
            }
            description
              "Value of the protocol set.";
            //Check if we can reuse an IANA-maintained module
          }
        }
      }
      container icmp-type-sets {
        description
          "Data definitions for a list of ICMP types which can be
           matched in policies.";
        list icmp-type-set {
          key "name";
          description
            "List of ICMP type set definitions.";
          leaf name {
            type string;
            description
              "Name of the ICMP type set -- this is used as a label to
               reference the set in match conditions.";
          }
          list types {
            key "type";
            description
              "Includes a list of ICMP types.";
            uses packet-fields:acl-icmp-header-fields;
          }
        }
      }
    }
    container aliases {
      description
        "Top-levl container for aliases.";
      list alias {
        key "name";
        description
          "List of aliases.";
        leaf name {
          type string;
          description
            "The name of the alias.";
        }
        uses alias;
      }
    }
  }
```

<CODE ENDS>

6. Security Considerations

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

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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

* TBC

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

* TBC

7. IANA Considerations

7.1. URI Registration

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

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

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

7.2. YANG Module Name Registration

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

```
name: ietf-acl-enh
namespace: urn:ietf:params:xml:ns:yang:ietf-acl-enh
maintained by IANA: N
prefix: acl-enh
reference: RFC XXXX

name: ietf-icmp-types
namespace: urn:ietf:params:xml:ns:yang:iana-icmp-types
maintained by IANA: Y
prefix: iana-icmp-types
reference: RFC XXXX
```

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <https://www.rfc-editor.org/rfc/rfc3032>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

[RFC5462] Andersson, L. and R. Asati, "Multiprotocol Label Switching (MPLS) Label Stack Entry: "EXP" Field Renamed to "Traffic Class" Field", RFC 5462, DOI 10.17487/RFC5462, February 2009, <https://www.rfc-editor.org/rfc/rfc5462>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/rfc/rfc6242>.

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

[RFC7209] Sajassi, A., Aggarwal, R., Uttaro, J., Bitar, N., Henderickx, W., and A. Isaac, "Requirements for Ethernet VPN (EVPN)", RFC 7209, DOI 10.17487/RFC7209, May 2014, <https://www.rfc-editor.org/rfc/rfc7209>.

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

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

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

[RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <https://www.rfc-editor.org/rfc/rfc8294>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

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

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

[RFC8956] Loibl, C., Ed., Raszuk, R., Ed., and S. Hares, Ed., "Dissemination of Flow Specification Rules for IPv6", RFC 8956, DOI 10.17487/RFC8956, December 2020, <https://www.rfc-editor.org/rfc/rfc8956>.

8.2. Informative References

[I-D.boucadair-netmod-iana-registries] Boucadair, M., "Recommendations for Creating IANA-Maintained YANG Modules", Work in Progress, Internet-Draft, draft-boucadair-netmod-iana-registries-07, 20 January 2023, <https://datatracker.ietf.org/doc/html/draft-boucadair-netmod-iana-registries-07>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/rfc/rfc8340>.

[RFC8955] Loibl, C., Hares, S., Raszuk, R., McPherson, D., and M. Bacher, "Dissemination of Flow Specification Rules", RFC 8955, DOI 10.17487/RFC8955, December 2020, <https://www.rfc-editor.org/rfc/rfc8955>.

[RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <https://www.rfc-editor.org/rfc/rfc9132>.

Appendix A. XLTS Template to Generate The ICMP Type IANA-Maintained Module

```
<CODE BEGINS>
<?xml version="1.0" encoding="utf-8"?>
<stylesheet
    xmlns="http://www.w3.org/1999/XSL/Transform"
    xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns:iana="http://www.iana.org/assignments"
    xmlns:yin="urn:ietf:params:xml:ns:yang:yin:1"
    version="1.0">
  <import href="../../../xslt/iana-yinx.xsl"/>
  <output method="xml" encoding="utf-8"/>
  <strip-space elements="*"/>
```

```
<template match="iana:registry[@id='icmp-parameters-types']">
  <element name="yin:typedef">
    <attribute name="name">icmp-type-name</attribute>
    <element name="yin:type">
      <attribute name="name">enumeration</attribute>
      <apply-templates
          select="iana:record[not(iana:description = 'Unassigned' or
                  starts-with(iana:description, 'Reserved') or
                  starts-with(iana:description, 'RFC3692')) or
                  contains(iana:description, 'experimental')]"/>
    </element>
    <element name="yin:description">
      <element name="yin:text">
        This enumeration type defines mnemonic names and
        corresponding numeric values of ICMP types.
      </element>
    </element>
    <element name="yin:reference">
      <element name="yin:text">
        RFC 2708: IANA Allocation Guidelines For Values In
                  the Internet Protocol and Related Headers
      </element>
    </element>
  </element>
  <element name="yin:typedef">
    <attribute name="name">icmp-type</attribute>
    <element name="yin:type">
      <attribute name="name">union</attribute>
      <element name="yin:type">
        <attribute name="name">uint8/</attribute>
      </element>
      <element name="yin:type">
        <attribute name="name">icmp-type-name</attribute>
      </element>
    </element>
    <element name="yin:description">
      <element name="yin:text">
        This type allows reference to an ICMP type using either
        the assigned mnemonic name or numeric value.
      </element>
    </element>
  </element>
</template>

<template match="iana:record">
  <call-template name="enum">
    <with-param name="id">
      <choose>
```

```
              <when test="contains(iana:description, '(Deprecated)')">
                <value-of select="translate(normalize-space(substring-before(iana:
description,
                      '(Deprecated)')),' ','')"/>
              </when>
              <otherwise>
                <value-of select="translate(normalize-space(iana:description),' ',
'')"/>
              </otherwise>
            </choose>
          </with-param>
          <with-param name="deprecated"
                      select="contains(iana:description, '(Deprecated)')"/>
        </call-template>
      </template>

   </stylesheet>
   <CODE ENDS>
```

Appendix B. Initial Version of the The ICMP Type IANA-Maintained Module

```
   <CODE BEGINS>
    file iana-icmp-types@2020-09-25.yang

   module iana-icmp-types {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:iana-icmp-types";
     prefix iana-icmp-types;

     organization
       "Internet Assigned Numbers Authority (IANA)";

     contact
       "Internet Assigned Numbers Authority

        ICANN
        12025 Waterfront Drive, Suite 300
        Los Angeles, CA 90094


        Tel: +1 424 254 5300

        <mailto:iana@iana.org>";

     description
       "This YANG module translates IANA registry 'ICMP Type Numbers' to
        YANG derived types.

        Copyright (c) 2023 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 was generated from the
        corresponding IANA registry using an XSLT stylesheet from the
        'iana-yang' project (https://github.com/llhotka/iana-yang).";

    reference
      "Internet Control Message Protocol (ICMP) Parameters
       (https://www.iana.org/assignments/icmp-parameters/)";

    revision 2020-09-25 {
      description
        "Current revision as of the revision date specified in the XML
         representation of the registry page.";
      reference
        "https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xml";
    }

    /* Typedefs */

    typedef icmp-type-name {
      type enumeration {
        enum EchoReply {
          value 0;
          description
            "Echo Reply";
          reference
            "RFC 792";
        }
        enum DestinationUnreachable {
          value 3;
          description
            "Destination Unreachable";
          reference
            "RFC 792";
        }
        enum SourceQuench {
          value 4;
          status deprecated;
          description
            "Source Quench (Deprecated)";
          reference
            "- RFC 792
```

```
            - RFC 6633";
        }
        enum Redirect {
          value 5;
          description
            "Redirect";
          reference
            "RFC 792";
        }
        enum AlternateHostAddress {
          value 6;
          status deprecated;
          description
            "Alternate Host Address (Deprecated)";
          reference
            "RFC 6918";
        }
        enum Echo {
          value 8;
          description
            "Echo";
          reference
            "RFC 792";
        }
        enum RouterAdvertisement {
          value 9;
          description
            "Router Advertisement";
          reference
            "RFC 1256";
        }
        enum RouterSolicitation {
          value 10;
          description
            "Router Solicitation";
          reference
            "RFC 1256";
        }
        enum TimeExceeded {
          value 11;
          description
            "Time Exceeded";
          reference
            "RFC 792";
        }
        enum ParameterProblem {
          value 12;
          description
```

```
            "Parameter Problem";
          reference
            "RFC 792";
        }
        enum Timestamp {
          value 13;
          description
            "Timestamp";
          reference
            "RFC 792";
        }
        enum TimestampReply {
          value 14;
          description
            "Timestamp Reply";
          reference
            "RFC 792";
        }
        enum InformationRequest {
          value 15;
          status deprecated;
          description
            "Information Request (Deprecated)";
          reference
            "- RFC 792
             - RFC 6918";
        }
        enum InformationReply {
          value 16;
          status deprecated;
          description
            "Information Reply (Deprecated)";
          reference
            "- RFC 792
             - RFC 6918";
        }
        enum AddressMaskRequest {
          value 17;
          status deprecated;
          description
            "Address Mask Request (Deprecated)";
          reference
            "- RFC 950
             - RFC 6918";
        }
        enum AddressMaskReply {
          value 18;
          status deprecated;
```

```
          description
            "Address Mask Reply (Deprecated)";
          reference
            "- RFC 950
             - RFC 6918";
        }
        enum Traceroute {
          value 30;
          status deprecated;
          description
            "Traceroute (Deprecated)";
          reference
            "- RFC 1393
             - RFC 6918";
        }
        enum DatagramConversionError {
          value 31;
          status deprecated;
          description
            "Datagram Conversion Error (Deprecated)";
          reference
            "- RFC 1475
             - RFC 6918";
        }
        enum MobileHostRedirect {
          value 32;
          status deprecated;
          description
            "Mobile Host Redirect (Deprecated)";
          reference
            "- David Johnson <>
             - RFC 6918";
        }
        enum IPv6Where-Are-You {
          value 33;
          status deprecated;
          description
            "IPv6 Where-Are-You (Deprecated)";
          reference
            "- Bill Simpson <mailto:Bill.Simpson&um.cc.umich.edu>
             - RFC 6918";
        }
        enum IPv6I-Am-Here {
          value 34;
          status deprecated;
          description
            "IPv6 I-Am-Here (Deprecated)";
          reference
```

```
            "- Bill Simpson <mailto:Bill.Simpson&um.cc.umich.edu>
             - RFC 6918";
          }
          enum MobileRegistrationRequest {
            value 35;
            status deprecated;
            description
              "Mobile Registration Request (Deprecated)";
            reference
              "- Bill Simpson <mailto:Bill.Simpson&um.cc.umich.edu>
               - RFC 6918";
          }
          enum MobileRegistrationReply {
            value 36;
            status deprecated;
            description
              "Mobile Registration Reply (Deprecated)";
            reference
              "- Bill Simpson <mailto:Bill.Simpson&um.cc.umich.edu>
               - RFC 6918";
          }
          enum DomainNameRequest {
            value 37;
            status deprecated;
            description
              "Domain Name Request (Deprecated)";
            reference
              "- RFC 1788
               - RFC 6918";
          }
          enum DomainNameReply {
            value 38;
            status deprecated;
            description
              "Domain Name Reply (Deprecated)";
            reference
              "- RFC 1788
               - RFC 6918";
          }
          enum SKIP {
            value 39;
            status deprecated;
            description
              "SKIP (Deprecated)";
            reference
              "- Tom Markson <mailto:markson&osmosys.incog.com>
               - RFC 6918";
          }
```

```
        enum Photuris {
          value 40;
          description
            "Photuris";
          reference
            "RFC 2521";
        }
        enum ICMPmessagesutilizedbyexperimentalmobilityprotocolssuchasSeamoby {
          value 41;
          description
            "ICMP messages utilized by experimental mobility protocols
             such as Seamoby";
          reference
            "RFC 4065";
        }
        enum ExtendedEchoRequest {
          value 42;
          description
            "Extended Echo Request";
          reference
            "RFC 8335";
        }
        enum ExtendedEchoReply {
          value 43;
          description
            "Extended Echo Reply";
          reference
            "RFC 8335";
        }
      }
      description
        "This enumeration type defines mnemonic names and corresponding
         numeric values of ICMP types.";
      reference
        "RFC 2708: IANA Allocation Guidelines For Values In the
         Internet Protocol and Related Headers";
    }

    typedef icmp-type {
      type union {
        type uint8;
        type icmp-type-name;
      }
      description
        "This type allows reference to an ICMP type using either the
         assigned mnemonic name or numeric value.";
    }
  }
```

<CODE ENDS>

Appendix C. Acknowledgements

Many thanks to Jon Shallow and Miguel Cros for the review and comments to the document, including prior to publishing the document.

Thanks to Qiufang Ma and Victor Lopez for the comments and suggestions.

The IANA-maintained model was generated using an XSLT stylesheet from the 'iana-yang' project (https://github.com/llhotka/iana-yang).

This work is partially supported by the European Commission under Horizon 2020 Secured autonomic traffic management for a Tera of SDN flows (Teraflow) project (grant agreement number 101015857).

Authors' Addresses

Oscar Gonzalez de Dios
Telefonica
Email: oscar.gonzalezdedios@telefonica.com

Samier Barguil
Telefonica
Email: samier.barguilgiraldo.ext@telefonica.com

Mohamed Boucadair
Orange
Email: mohamed.boucadair@orange.com

Qin Wu
Huawei
Email: bill.wu@huawei.com

NETMOD Working Group
Internet-Draft
Updates: 8407 (if approved)
Intended status: Standards Track
Expires: 9 January 2024

Q. Wu
B. Claise
Huawei
M. Boucadair
Orange
P. Liu
Z. Du
China Mobile

8 July 2023

Node Tags in YANG Modules
draft-ietf-netmod-node-tags-10

Abstract

This document defines a method to tag nodes that are associated with the 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 9 January 2024.

Copyright Notice

Table of Contents

1. Introduction

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

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 these tags can be associated with nodes within a YANG module, to:

* 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 which add node tags or from which to remove node tags, as well as a way 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 [RFC6022] 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 are assigned during the module definition time are added;

2. User-configured tags (i.e., tags of "intended" origin) that are dynamically defined and added 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 values of YANG metadata annotation are 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 6 are retrieved centrally via the "ietf-node-tags" module and can be either assigned during the module defintion time or dynamically set by the client for a given data node instance.

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.

Metric: Metrics are a specific kind of telemetry data. They represent a snapshot of the current state for a set of data. They are distinct from logs or events, which focus on records or information about individual events [OpenMetrics].

Info: Info is used to expose textual information which SHOULD NOT change during process lifetime. Common examples are an application's version [OpenMetrics].

Gauge: Gauges are current measurements, such as bytes of memory currently used or the number of items in a queue. For gauges the absolute value is what is of interest to a user [OpenMetrics].

Counter: Counters measure discrete events. Common examples are the number of HTTP requests received, CPU seconds spent, or bytes sent. For counters how quickly they are increasing over time is what is of interest to a user [OpenMetrics].

Summary: Summaries measure distributions of discrete events and can be used to measure an average event size [OpenMetrics].

Unknown: Unknown MAY be used when it is impossible to determine the types of individual metrics from 3rd party systems [OpenMetrics].

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

3. Sample Use Cases for Node Tags

The following describes some 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 node 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. All tag prefixes MUST end with a colon and Colons MUST NOT be used within a prefix. An IANA registry (Section 9.1) is used to register node tag prefixes. Three prefixes are defined in the subsections that follow.

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. These IETF Node Tags MUST conform to Net-Unicode as defined in [RFC5198], and SHOULD not need normalization.

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) [RFC9371].

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.

Therefore an implementation SHOULD be able to process all tags regardless of their prefixes.

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 that wishes to define additional tags to associate with data nodes within a YANG module MAY do so at implementation time. These tags SHOULD be IETF (i.e., registered)), but MAY be vendor tags. IETF tags allows better interoperability than vendor tags.

5.3. User Tagging

Node tags that are dynamically defined, with or without a prefix, can be added 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 shown as figure 1:

```
module: ietf-node-tags
augment /tags:module-tags/tags:module:
  +--rw node-tags
     +--rw node* [id]
        +--rw id            unit64
        +--rw node-selector nacm:node-instance-identifier
        +--rw tags*         tags:tag
        +--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: Mohamed Boucadair
            <mailto:mohamed.boucadair@orange.com>
```

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

       Editor: Zongpeng Du
               <mailto:duzongpeng@chinamobile.com>";
      // RFC Ed.: replace XXXX with actual RFC number and
      // remove this note.
     description
       "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";
     }
     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 uint64;
            description
              "Identification of each data node within YANG module. It is
               unique 64-bit unsigned integers.";
          }
          leaf node-selector {
            type nacm:node-instance-identifier;
            description
              "Selects the data nodes for which tags are specified.";
          }
         leaf-list tags {
            type tags: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-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 nodes 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:metric";
      }
      leaf bar {
        ntags:node-tag "ietf:info";
      }
    }
  }
  // ...
}
```

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 of RFC xxxx.

A data node can contain one or multiple node tags. Not all data nodes need to be tagged. A data node to be tagged with an initial value from Table 2 can be one of 'container', 'leaf-list', 'list', or 'leaf'. The 'container','leaf-list','list', or 'leaf' node not representing a snapshot of the current state for a set of data MUST not be tagged. The notification and action nodes MUST not be tagged.

All tag values described in Table 2 can be inherited down the containment hierarchy if the data nodes tagged with those tag values is one of 'container', 'leaf-list', or 'list'.

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:

Prefix	Description	Reference	Assignee
ietf:	IETF Tags allocated in the IANA IETF YANG Node Tags registry	[This document]	IETF
vendor:	Non-registered tags allocated by the module's implementer.	[This document]	IETF
user:	Non-registered tags allocated by and for the user.	[This document]	IETF

Figure 2: Table 1

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 with Expert 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:

Node Tag	Description	Reference
ietf:metric	Represent metric data (e.g., ifstatistics) associated with specific node (e.g.,interfaces)	[This document] [Open Metrics]
ietf:info	Represent texture info (e.g., software revision) associated with specific node (e.g.,component)	[This document] [Open Metrics]
ietf:delay	Represent the delay metric data associated with specific node.	[This document [RFC2681] [RFC7679]
ietf:jitter	Represent the jitter metric data asociated with specific node.	[This document] [RFC3393]
ietf:loss	Represent the loss metric data associated with specific node.	[This document] [RFC7680]

		[RFC6673]
ietf:counter	Represent any metric value associated with specific node that monotonically increases over time, starting from zero.	[This document] [Open Metrics]
ietf:gauge	Represent current measurements associated with specific node that may increase, decrease or stay constant.	[This document] [Open Metrics]
ietf:summary	Represent the metric value associated with specific node that measures distributions of discrete events without knowing predefined range.	[This document] [Open Metrics]
ietf:unknown	Represent the metric value associated with specific node that can not determine the type of metric.	[This document] [Open Metrics]

Figure 3: Table 2

9.3. Updates to the IETF XML Registry

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

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

URI: urn:ietf:params:xml:ns:yang:ietf-node-tags-state
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 two YANG modules in the YANG Module Names registry [RFC6020] within the "YANG Parameters" registry:

```
name: ietf-node-tags
namespace: urn:ietf:params:xml:ns:yang:ietf-node-tags
prefix: ntags
reference: RFC XXXX

name: ietf-node-tags-state
namespace: urn:ietf:params:xml:ns:yang:ietf-node-tags-state
prefix: ntags-s
reference: RFC XXXX
```

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

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

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

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

[RFC8819] Hopps, C., Berger, L., and D. Bogdanovic, "YANG Module Tags", RFC 8819, DOI 10.17487/RFC8819, January 2021, <https://www.rfc-editor.org/info/rfc8819>.

13.2. Informative References

[FCAPS] International Telecommunication Union, "X.700 : Management framework for Open Systems Interconnection (OSI) for CCITT applications", , September 1992, <http://www.itu.int/rec/T-REC-X.700-199209-I/en>.

[OpenMetrics] OpenMetrics, "OpenMetrics, a cloud-native, highly scalable metrics protocol", , 31 March 2022, <https://github.com/OpenObservability/OpenMetrics/blob/main/specification/OpenMetrics.md>.

[RFC6022] Scott, M. and M. Bjorklund, "YANG Module for NETCONF Monitoring", RFC 6022, DOI 10.17487/RFC6022, October 2010, <https://www.rfc-editor.org/info/rfc6022>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

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

[RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <https://www.rfc-editor.org/info/rfc8526>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[RFC9195] Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", RFC 9195, DOI 10.17487/RFC9195, February 2022, <https://www.rfc-editor.org/info/rfc9195>.

[RFC9196] Lengyel, B., Clemm, A., and B. Claise, "YANG Modules Describing Capabilities for Systems and Datastore Update Notifications", RFC 9196, DOI 10.17487/RFC9196, February 2022, <https://www.rfc-editor.org/info/rfc9196>.

[RFC9371] Baber, A. and P. Hoffman, "Registration Procedures for Private Enterprise Numbers (PENs)", RFC 9371, DOI 10.17487/RFC9371, March 2023, <https://www.rfc-editor.org/info/rfc9371>.

Appendix A. 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:

```
module: example-tunnel-pm
        +--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 [RFC9195][RFC9196] are created as follows:

```
    <rpc message-id="103"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <edit-data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda"
                 xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        <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>1743</id>
            <node-selector>/tp:tunnel-svc[name='foo']/tp:packet-loss
             /</name>
            <tag>user:customer1_example_com</tag>
            <tag>user:critical</tag>
          </node>
          <node>
            <id>1744</id>
            <node-selector>/tp:tunnel-svc[name='bar']/tp:modified-time
            /</node-selctor>
            <tag>user:customer2_example_com</tag>
          </node>
        </node-tags>
       </module>
       </module-tag>
        </config>
      </edit-data>
    </rpc>
```

Note that the 'user:critical' tag is one addtional new tag value.

Appendix B. 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].

```
 =============== NOTE: '\' line wrapping per RFC 8792 ================

 <ns0:data xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0">
  <t:module-tags xmlns:t="urn:ietf:params:xml:ns:yang:ietf-module-tags">
     <t:module>
        <t:name>ietf-interfaces</t:name>
        <s:node-tags
          xmlns:s="urn:ietf:params:xml:ns:yang:ietf-node-tags">
         <s:node>
          <s:id>1723</s:id>
          <s:node-selector>
           /if:interfaces/if:interface/if:statistics/if:in-errors
          </s:node-selector>
          <s:tag>ietf:metric</s:tag>
          <s:tag>ietf:loss</s:tag>
          <s:tag>vendor:agg</s:tag>
         </s:node>
       </s:node-tags>
     </t:module>
     <t:module>
        <t:name>ietf-ip</t:name>
        <s:node-tags
          xmlns:s="urn:ietf:params:xml:ns:yang:ietf-node-tags">
         <s:node>
         <s:id>1733</s:id>
         <s:node-selector>/if:interfaces/if:interface/ip:ipv4/ip:mtu
         </s:node-selector>
         <s:tag>ietf:metric</s:tag>
        </s:node>
       </s:node-tags>
     </t:module>
   </t:module-tags>
 </ns0:data>
```

Figure 4: Example NETCONF Query Output

Appendix C. 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;
```

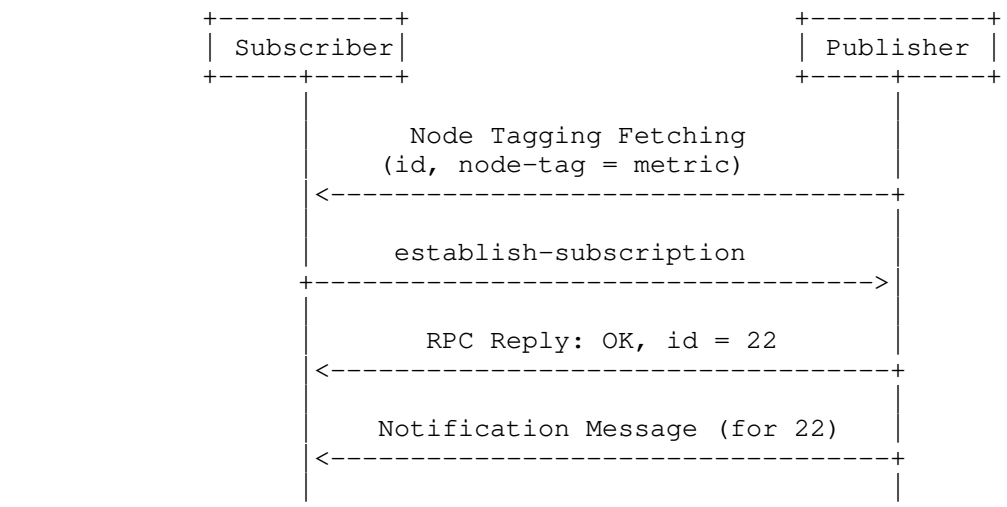
```
     import ietf-netconf-acm {
       prefix nacm;
       reference
       "RFC 8341: Network Configuration Access Control
                  Model";
     }
     import ietf-module-tags {
       prefix tags;
     }
     import ietf-module-tags-state {
       prefix tags-s;
       reference
       "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: Mohamed Boucadair
               <mailto:mohamed.boucadair@orange.com>

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

       Editor: Zongpeng Du
               <mailto:duzongpeng@chinamobile.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";
      }
      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 uint64;
              description
              "Identification of each data node within YANG module. It is
               unique 64-bit unsigned integers.";
            }
            leaf node-selctor {
              type nacm:node-instance-identifier;
              mandatory true;
              status deprecated;
              description
                "Selects the data nodes for which tags are
                 specified.";
            }
           leaf-list tags {
```

```
            type tags: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-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
               operational effect.";
           }
         }
       }
     }
   }
   <CODE ENDS>
```

Appendix D. 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.



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] [RFC8641].

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<?xml version="1.0" encoding="UTF-8"?>
    <t:module-tags
      xmlns:t="urn:ietf:params:xml:ns:yang:ietf-module-tags">
       <t:module>
          <t:name>ietf-interfaces</t:name>
          <s:node-tags
            xmlns:s="urn:ietf:params:xml:ns:yang:ietf-node-tags">
           <s:node>
            <s:id>1723</s:id>
            <s:node-selector>/if:interfaces/if:interface/if:in-errors
            /</s:node-selector>
            <s:tag>ietf:metric</s:tag>
            <s:tag>ietf:loss</s:tag>
           </s:node>
           </s:node-tags>
      </t:module>
    </module-tags>
```

Figure 5: List of Available Target Objects

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]and [RFC8641] 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:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

 <netconf:rpc message-id="101"
     xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
   <establish-subscription
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifica\
       tions"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
     <yp:datastore
         xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
       ds:operational
     </yp:datastore>
     <yp:datastore-xpath-filter
        xmlns:ex="https://example.com/sample-data/1.0">
       /if:interfaces/if:interface/if:statistics/if:in-errors
     </yp:datastore-xpath-filter>
     <yp:periodic>
       <yp:period>500</yp:period>
     </yp:periodic>
   </establish-subscription>
 </netconf:rpc>
```

Appendix E. Changes between Revisions

Editorial Note (To be removed by RFC Editor)

v09 - v10

* Remove identityref type from YANG module to avoid duplciation with IETF node tag and align with Module tag design in RFC 8819.

* Add one key leaf using unsigned integer type to identify each data node and modify the id leaf into path leaf.

* Clarify the colon's meaning and how it is used in the node tags.

* Remove Appendix A and Update Appendix B to explain how additonal tags can be added at the implementation time.

* Module structure changes and YANG module code changes to align with Module tag design in RFC 8819.

* Add relevant RFCs referencing to IETF node tags defined in section 9.2 and provide additional term definition to support IETF node tags defined in section 9.2.

* Specify which data nodes can be tagged, which data nodes can not in section 8.1.

v08 - v09

* Clarification on the relation with metadata annotation in section 1.

* Clarification on how masked-tag is used in section 5.3.

* Other editorial changes.

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 be 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 exlusive.

* 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

Mohamed Boucadair
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.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

NETMOD
Internet-Draft
Updates: 8342, 6241, 8526, 8040 (if approved)
Intended status: Standards Track
Expires: 5 January 2024

Q. Ma, Ed.
Q. Wu
C. Feng
Huawei

4 July 2023

System-defined Configuration
draft-ietf-netmod-system-config-02

Abstract

This document describes how a management client and server handle YANG-modeled configuration data that is defined by the server itself. The system-defined configuration can be referenced (e.g. leafref) by configuration explicitly created by a client.

The Network Management Datastore Architecture (NMDA) defined in RFC 8342 is updated with a read-only conventional configuration datastore called "system" to hold system-defined configuration. As an alternative to clients explicitly copying referenced system-defined configuration into the target configuration datastore (e.g., <running>) so that the datastore is valid, a "resolve-system" parameter is defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. This solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay (e.g., copy system configuration using the same key value as in <system>) and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

This document updates RFC 8342, RFC 6241, RFC 8526 and RFC 8040.

This Internet-Draft will expire on 5 January 2024.

Table of Contents

1. Introduction

The Network Management Datastore Architecture (NMDA) [RFC8342] defines system configuration as the configuration that is supplied by the device itself and appears in <operational> when it is in use (Figure 2 in [RFC8342]).

However, there is a desire to enable a server to better structure and expose the system configuration. NETCONF/RESTCONF clients can benefit from a standard mechanism to retrieve what system configuration is available on a server.

Some servers allow the NETCONF/RESTCONF client to reference a system-defined node which isn't present in the target datastore (e.g., <running>). The absence of the system configuration in the datastore can render the datastore invalid from the perspective of a client or offline tools (e.g., missing leafref targets). This document describes several approaches to bring the datastore to a valid state and ensuing that all referential integrity constraints are satisfied.

Some servers allow the descendant nodes of system-defined configuration to be configured or modified. For example, the system configuration may contain an almost empty physical interface, while the client needs to be able to add, modify, or 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 the Network Management Datastore Architecture (NMDA) defined in RFC 8342 with a read-only conventional configuration datastore called "system" to hold system-defined configuration. As an alternative to clients explicitly copying referenced system-defined configuration into the target configuration datastore (e.g., <running>) so that the datastore is valid, a "resolve-system" parameter has been defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. This solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay (e.g., copy system configuration using the same key value as in <system>) and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

If a system-defined node is referenced, it refers to one of the following cases throughout this document:

* It is present in a leafref "path" statement and referred as the leafref value

* It is used as an "instance-identifier" type value

* It is present in an Xpath expression of "when" or "must" constraints

* It is defined to satisfy the "mandatory" constraints

* It is defined to exactly satisfy the "min-element" constraints

Conformance to this document requires the NMDA servers to implement the "ietf-system-datastore" YANG module (Section 6).

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:

System configuration: Configuration that is provided by the system itself. System configuration is present in the system configuration datastore (regardless of being applied by the device or referenced by other configuration nodes), and appears in the intended configuration datastore. System configuration that is considered active (according to the NMDA defined in RFC 8342) appears in <operational> with origin="system". It is a different and separate concept from factory default configuration defined in RFC 8808 (which represents a preset initial configuration that is used to initialize the configuration of a server).

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

This document redefines the term "conventional configuration datastore" in Section 3 of [RFC8342] 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> are read-only to clients but may change dynamically. <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 and 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.

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, regardless of which target datastore it is.

1.5. Updates to RFC 8040

This document extends Sections 4.8 and 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.

Name	Methods	Description
resolve-system	POST, PUT PATCH	resolve any references not resolved by the client and copy referenced system configuration into <running> automatically. This parameter can be given in any order.

Figure 1: RESTCONF "resolve-system" Query Parameter

1.5.2. Query Parameter URI

To enable a 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

Comment: Should we define a similar capability identifier for NETCONF protocol?

2. Kinds of System Configuration

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

Active system configuration refers to configuration that is in use by a device. As per definition of the operational state datastore in [RFC8342], if system configuration is inactive, it should not appear in <operational>. However, system configuration is present in <system> once it is generated, regardless of whether it is active or not.

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 loopback 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 a QoS feature is enabled, related QoS policies are automatically created by the system.

2.3. Inactive-Until-Referenced

There are some system configurations predefined (e.g., application ids, anti-x signatures, trust anchor certs, etc.) as a convenience for the clients, which must be referenced to be active. The clients can also define their own configurations for their unique requirements. Inactive-until-referenced system configurations are generated in <system> immediately when the device is powered on, but they are not active until being referenced.

3. The System Configuration Datastore (<system>)

NMDA servers compliant with this document MUST implement a system configuration datastore, and they SHOULD also implement <intended>.

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.

* 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 protocols such as NETCONF, RESTCONF, but may change itself by upgrades and/or when resource-conditions are met. The datastore can be read using the standard network management protocols such as NETCONF and RESCTCONF.

* Origin: This document does not define any new origin identity when it interacts with <intended> 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>. 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 configuration 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.

4. Static Characteristics of <system>

4.1. Read-only to Clients

The system 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 5.4).

4.2. 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 a QoS feature is enabled, QoS-related policies are created by the system.

If the system configuration gets changed, YANG notifications (e.g., "push-change-update" notification) [RFC6470][RFC8639][RFC8641] 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.

4.3. No Impact to <operational>

This work intends to have no impact to <operational>. System configuration appears in <operational> with "origin=system". This document 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.

5. Dynamic Behavior

5.1. Conceptual Model of Datastores

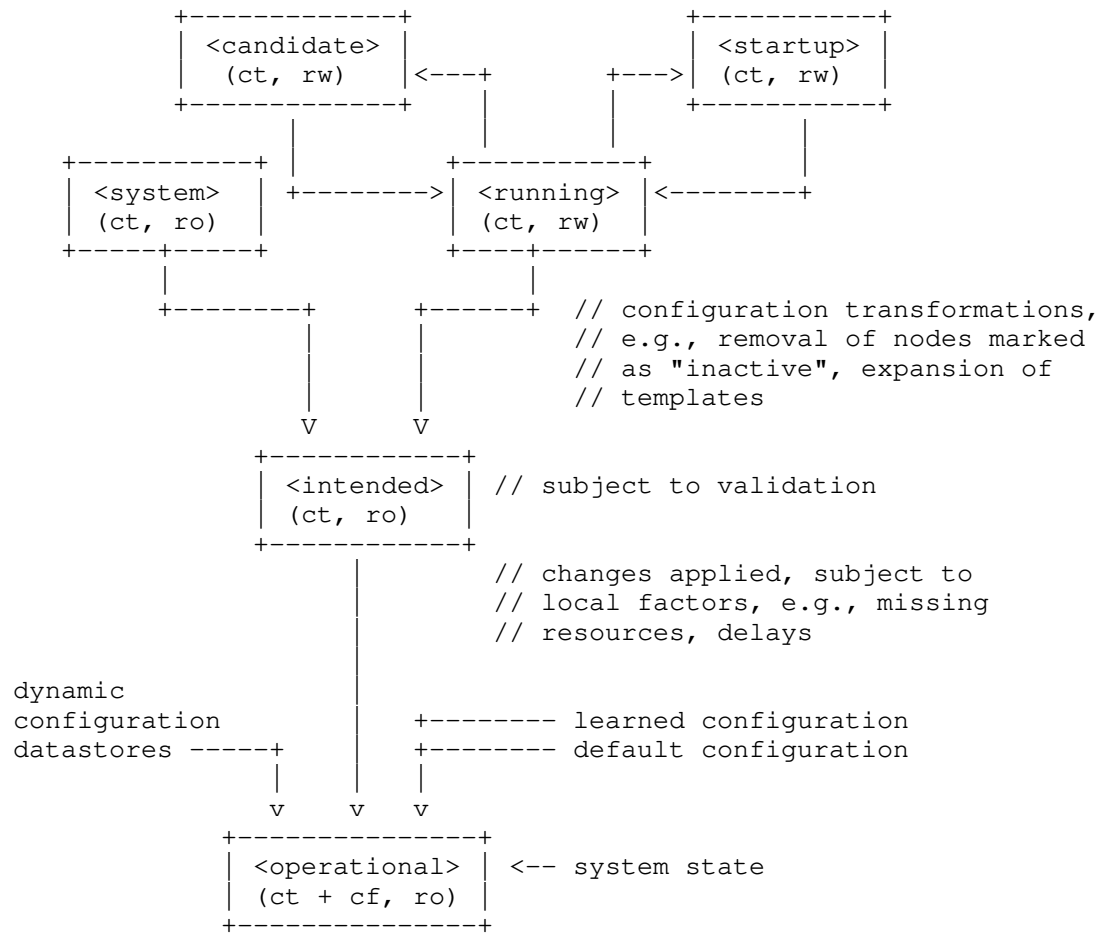
This document introduces a datastore named "system" which is used to hold all three types of system configurations defined in Section 2.

When the device is powered on, immediately-active system configuration will be generated in <system> and active immediately, but inactive-until-referenced system configuration only becomes active if it is referenced by client-defined configuration. While conditionally-active system configuration will only be created and active 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> takes precedence over the same node in <system> 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>.

As a result, Figure 2 in Section 5 of RFC 8342 is updated with the below conceptual model of datastores which incorporates the system configuration datastore.



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

Figure 2: Architectural Model of Datastores

Servers MUST enforce that configuration references in <running> are resolved within <running> and ensure that <running> contains any referenced system configuration. 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 configurations 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.

Configuration defined in <system> is merged into <intended>. It is also present in <operational> if it is in use by the device, even if a client may delete the configuration which is copied from <system> into <running>. For example, system initializes a value for a particular leaf which is overridden by the client with a different value in <running>. The client may delete that node in <running>, in which case system-initialized value defined in <system> can be still in use and appear in <operational>.

Applied system configuration regardless of explicitly or automatically being copied into <running>, appears in <operational> with origin="system".

Comment: this might need further discussion: should the origin="system" be required for system configuration copied/pasted into <running>?

Any deletable system-provided configuration that is placed into <running> by the system at boot up, without being part of the contents of a <startup> datastore, 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.

5.2. 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>.

The explicit configuration of system-defined nodes in the target datastore (e.g., <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 but need the datastore to be valid. The client can explicitly declare (i.e., configure in the datastore like <running>) the list entries (with at least the keys) for any system configuration list entries that are referenced elsewhere in <running>. The client does not necessarily need to declare all the contents of the list entry (i.e. the descendant nodes) , only the parts that are required to make the datastore appear valid.

5.3. Servers Auto-configuring Referenced System Configuration ("resolve-system" parameter)

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, and the server supports this capability, 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.

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 <running> valid.

There is no distinction between the configuration in the target datastore (e.g., <running>) which is automatically configured by the server and the one explicitly declared by the client, e.g., a read back of the datastore (i.e., <get>, <get-config> or <get-data> operation) returns automatically configured nodes. Note that even an auto-configured node is allowed to be deleted from the target datastore by the client, the operation request (e.g., <edit-config>) may not succeed due to incomplete referential integrity, it is also possible that the system automatically configures the deleted node again to make configuration valid, when a "resolve-system" parameter is carried. A referenced system node onced auto-configured in the datastore, will not be removed or updated automatically by the server even in cases like all references to it are deleted by the client or system configuration is no longer present in <system> due to factors like device upgrade or system-controlled resources (e.g., HW unavailable) change.

Comment: Should the server update configuration in <running> that is copied from <system> automatically (and manually?) during an upgrade? Jason: I think maybe servers that convert configuration during upgrade (a common approach) would want to convert/upgrade system config as well as any copied system config that exists in running.

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. Not using capitalized "SHOULD NOT" in the previous sentence is intentional. The intention is to bring awareness to the general need to not surprise clients with unexpected changes. It is desirable for clients to always opt into using mechanisms having server-side changes. This document enables a client to opt into this behavior using the "resolve-system" parameter. An example of this type of opt-in behavior can also be found in RFC 7317, which enables a client to opt into its behavior using a "0" prefix (see ianach:crypt-hash type defined in [RFC7317]).

Support for the "resolve-system" parameter is OPTIONAL. Non-NMDA servers MAY also implement this parameter without implementing the system configuration datastore, which would only eliminate the ability to expose the system configuration via protocol operations. If a server implements <system>, referenced system configuration is copied from <system> into the target datastore(e.g., <running>) when the "resolve-system" parameter is used; otherwise it is an implementation decision where to copy referenced system configuration into the target datastore (e.g., <running>).

Comments from Jason: Overall the resolve-system function may mean an expensive (time consuming) operation on the server side. Conceptually it may mean doing a validation on the running, and then when an error is hit, searching the 'system' datastore for something that could resolve that invalid aspect. Then running validation again and hitting the next error. It may require multiple passes (since some errors are dependent on the previous error being present or 'fixed').

5.4. Modifying (Overriding) System Configuration

In some cases, a server may allow some parts of system configuration to be modified. 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.

For instance, 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 constraints and NACM rules, and other server-internal restrictions. If a system configuration node is non-modifiable, then writing a different value for that node MUST return an error. The immutability of system configuration is further defined in [I-D.ma-netmod-immutable-flag].

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).

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 <running> values go before or after, or is this a case where a full-replace is needed.

5.5. Examples

This section shows some 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.

5.5.1. 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 configurations are active 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 dest_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:

```
<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>
              <dest_address>192.0.2.0/24</dest_address>
            </ipv4>
            <application>ftp</application>
            <application>tftp</application>
            <application>my-app-1</application>
          </matches>
          <packet_action>forward</packet_action>
        </acl_rule>
      </acl>
    </config>
    <resolve-system/>
  </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>
            </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, and the server treats application "smtp" configuration as "inactive-until-referenced", it does not appear in <operational> but only in <system>.

5.5.2. 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>
              <dest_address>192.0.2.0/24</dest_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:

```
          <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>
            </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>
         <name>ftp</name>
         <protocol or:origin="or:system">tcp</protocol>
         <destination-port or:origin="or:system">21</destination-port>
       </application>
       <application>
         <name>tftp</name>
         <protocol or:origin="or:system">udp</protocol>
         <destination-port or:origin="or:system">69</destination-port>
       </application>
     </applications>
```

Since the application names "ftp" and "tftp" are explicitly configured by the client, they take precedence over the values in <system>, the "origin" attribute will be set to "intended".

5.5.3. Modifying a System-instantiated Leaf's Value

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

```
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>
```

5.5.4. 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:

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

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

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

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

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

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 configuration 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>:

```
{
    "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@2023-07-04.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 NETDOD (Network Modeling) Working Group";
  contact
    "WG Web:   https://datatracker.ietf.org/wg/netmod/
     WG List:  NETMOD WG list <mailto:netmod@ietf.org>

     Author: Qiufang Ma
             <mailto:maqiufang1@huawei.com>
     Author: Qin Wu
             <mailto:bill.wu@huawei.com>
```

```
        Author: Chong Feng
                <mailto:frank.fengchong@huawei.com>";
      description
        "This module defines a new YANG identity that uses the
         ds:datastore identity defined in [RFC8342].

         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 2023-07-04 {
        description
          "Initial version.";
        reference
          "RFC XXXX: System-defined Configuration";
      }

      identity system {
        base ds:conventional;
        description
          "This read-only datastore contains the 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" YANG module 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}?
    |     +---w resolve-system?      empty
    +---x copy-config
    |  +---w input
    |     +---w target
```

```
      |      |  +---w (config-target)
      |      |     +--:(candidate)
      |      |     |  +---w candidate?   empty {candidate}?
      |      |     +--:(running)
      |      |     |  +---w running?     empty {writable-running}?
      |      |     +--:(startup)
      |      |     |  +---w startup?     empty {startup}?
      |      |     +--:(url)
      |      |        +---w url?         inet:uri {url}?
      |      +---w source
      |      |  +---w (config-source)
      |      |     +--:(candidate)
      |      |     |  +---w candidate?   empty {candidate}?
      |      |     +--:(running)
      |      |     |  +---w running?     empty
      |      |     +--:(startup)
      |      |     |  +---w startup?     empty {startup}?
      |      |     +--:(url)
      |      |     |  +---w url?         inet:uri {url}?
      |      |     +--:(config)
      |      |        +---w config?      <anyxml>
      |      +---w resolve-system?       empty
      +---x edit-data
         +---w input
            +---w datastore            ds:datastore-ref
            +---w default-operation?   enumeration
            +---w (edit-content)
            |  +--:(config)
            |  |  +---w config?        <anydata>
            |  +--:(url)
            |     +---w url?           inet:uri {nc:url}?
            +---w resolve-system?      empty
```

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 configurations 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", <system> shows the following configuration of loopback interface:

```
   <interfaces xmlns="urn:example:interfacemgmt">
     <interface>
       <name>lo0</name>
       <mtu>1500</mtu>
       <ip-address>127.0.0.1</ip-address>
     </interface>
   </interfaces>
```

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@2023-07-04.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";
  }
```

```
  organization
    "IETF NETMOD (Network Modeling) 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: Chong Feng
             <mailto:frank.fengchong@huawei.com>";
  description
    "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) 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 2023-07-04 {
    description
      "Initial version.";
    reference
      "RFC XXXX: System-defined Configuration";
  }

  grouping resolve-system-grouping {
    description
```

```
        "Define the resolve-system parameter grouping.";
      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:edit-config/nc:input" {
      description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid.";
      uses resolve-system-grouping;
    }

    augment "/nc:copy-config/nc:input" {
      description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid.";
      uses resolve-system-grouping;
    }

    augment "/ncds:edit-data/ncds:input" {
      description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid.";
      uses resolve-system-grouping;
    }
  }

  <CODE ENDS>
```

8. IANA Considerations

8.1. The "IETF XML" Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-system-datastore
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-resolve-system
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
namespace: urn:ietf:params:xml:ns:yang:ietf-system-datatstore
maintained by IANA: N
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

name: ietf-netconf-resolve-system
prefix: ncrs
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-resolve-system
maintained by IANA: N
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]:

Index	Capability Identifier
:resolve-system	urn:ietf:params:restconf:capability:resolve-system:1.0

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

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net

Jan Lindblad
Cisco Systems

Email: jlindbla@cisco.com

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf@chinatelecom.cn

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Acknowledgements

The authors would like to thank for following for discussions and providing input to this document (ordered by first name): Alex Clemm, Andy Bierman, Balazs Lengyel, Juergen Schoenwaelder, Martin Bjorklund, Mohamed Boucadair, Robert Wilton and Timothy Carey.

References

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

[RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <https://www.rfc-editor.org/info/rfc8526>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

Informative References

[I-D.ma-netmod-immutable-flag]
Ma, Q., Wu, Q., Lengyel, B., and H. Li, "YANG Extension and Metadata Annotation for Immutable Flag", Work in Progress, Internet-Draft, draft-ma-netmod-immutable-flag-07, 25 May 2023, <https://datatracker.ietf.org/doc/html/draft-ma-netmod-immutable-flag-07>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

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

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

[RFC8808] Wu, Q., Lengyel, B., and Y. Niu, "A YANG Data Model for Factory Default Settings", RFC 8808, DOI 10.17487/RFC8808, August 2020, <https://www.rfc-editor.org/info/rfc8808>.

Appendix A. Key Use Cases

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

A.1. Device Powers On

<running>:

No configuration for "lo0" appears in <running>;

<system>:

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

<intended>:

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

   <operational>:

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

A.2. Client Commits Configuration

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

```
   <running>:

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

   <system>:

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

   <intended>:
```

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

```
   <operational>:

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

A.3. Operator Installs Card into a Chassis

```
   <running>:

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

```
   <system>:

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

```
   <intended>:

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

   <operational>:

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

Appendix B. Changes between Revisions

v01 - v02

* Define referenced system configuration
* better clarify "resolve-system" parameter
* update Figure 2 in NMDA RFC
* Editorial changes

v00 - v01

* Clarify why client's explicit copy is not preferred but cannot be avoided if resolve-system parameter is not defined

* Clarify active system configuration

* Update the timing when the server's auto copy should be enforced if a resolve-system parameter is used

* Editorial changes

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

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

Feng Chong
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: frank.fengchong@huawei.com

Network Working Group
Internet-Draft
Updates: 6020, 7950, 8407, 8525 (if approved)
Intended status: Standards Track
Expires: 19 October 2023

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

17 April 2023

Updated YANG Module Revision Handling
draft-ietf-netmod-yang-module-versioning-09

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 a minimum revision suggestion to help document 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 19 October 2023.

Table of Contents

1. Introduction

The current YANG [RFC7950] module update rules require that updates of YANG modules preserve strict backwards compatibility. This has caused problems as described in [I-D.ietf-netmod-yang-versioning-reqs]. This document recognizes the need to sometimes allow YANG modules to evolve with non-backwards-compatible changes, which can 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 when these changes occur, and to manage their effect on clients and the broader YANG ecosystem.

This document defines a flexible versioning solution. Several other documents build on this solution with additional capabilities. [I-D.ietf-netmod-yang-schema-comparison] specifies an algorithm that can be used to compare two revisions of a YANG schema and provide granular information to allow module users to determine if they are impacted by changes between the revisions. The [I-D.ietf-netmod-yang-semver] document extends the module versioning work by introducing a revision label scheme based on semantic versioning. YANG packages [I-D.ietf-netmod-yang-packages] provides a mechanism to group sets of related YANG modules together in order to manage schema and conformance of YANG modules as a cohesive set instead of individually. Finally, [I-D.ietf-netmod-yang-ver-selection] provides a schema selection mechanism that allows a client to choose which schemas to use when interacting with a server from the available schema that are

supported and advertised by the server. These other documents are mentioned here as informative references. Support of the other documents is not required in an implementation in order to take advantage of the mechanisms and functionality offered by this module versioning document.

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.

* Updated guidance for revision selection on imports and a YANG extension statement allowing YANG module imports to document 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 4.1 describes a YANG extension statement to describe potential YANG import revision dependencies.

This document updates [RFC7950] section 5.2, [RFC6020] section 5.2 and [RFC8407] section 3.2. 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 leafs 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.

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

* schema node

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.

* Backwards-compatible (BC) change: A backwards-compatible change between two YANG module revisions, as defined in Section 3.1.1

* Non-backwards-compatible (NBC) change: A non-backwards-compatible change between two YANG module revisions, as defined in Section 3.1.2

3. Refinements to YANG revision handling

[RFC7950] 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.

For a given YANG module revision, revision B is defined as being derived from revision A, if revision A is listed in the revision history of revision B. Although this document allows for a branched revision history, a given YANG module revision history does not contain all revisions in all possible branches, it only lists those from which is was derived, i.e., the module revision's history describes a single path of derived revisions back to the root of the module's revision history.

A corollary to the text above is that the ancestry (derived relationship) between two module or submodule revisions cannot be determined by comparing the module or submodule revision date or label alone - the revision history must be consulted.

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.

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.

Note that NBC changes often create problems for clients, thus it is recommended to avoid making them.

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 a non-backwards-compatible change.

* YANG schema nodes with a "status" "obsolete" substatement MAY be removed from published modules, and the removal is classified as a backwards-compatible change. 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.

* 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 change made to the "revision-date" or "recommended-min" substatements of an "import" statement, including adding new "revision-date" or "recommended-min" substatements, changing the argument of any "revision-date" or "recommended-min" substatetements, or removing any "revision-date" or "recommended-min" substatements, is classified as 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 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.

Adding, modifying or removing a "rev:non-backwards-compatible" extension statement is considered to be a BC change.

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 recommended-min 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:label 4.0.0;
  rev:non-backwards-compatible;
}

revision 2020-08-09 {
  rev:label 3.0.0;
  rev:non-backwards-compatible;
}

revision 2020-06-07 {
  rev:label 2.1.0;
}

revision 2020-02-10 {
  rev:label 2.0.0;
  rev:non-backwards-compatible;
}

revision 2019-10-21 {
  rev:label 1.1.3;
}

revision 2019-03-04 {
  rev:label 1.1.2;
}

revision 2019-01-02 {
  rev: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:recommended-min" 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, [RFC6020] section 5.2 and [RFC8407] section 3.2

If a revision has an associated revision label, then it is RECOMMENDED that the name of the file for that revision be of the form:

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

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

YANG module (or submodule) files may be identified using either the revision-date (as per [RFC8407] section 3.2) or the revision label.

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:

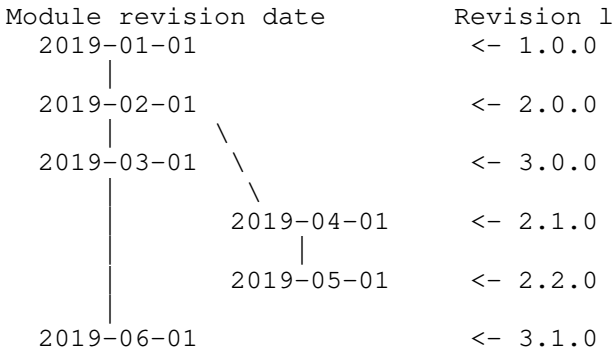
* specifies the revision label scheme used by the module or submodule
* is defined in the document which specifies the revision label scheme
* MUST be an identity derived from "revision-label-scheme-base".

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.



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:label 3.1.0;
    description "Add new functionality.";
  }

  revision 2019-03-01 {
    rev:label 3.0.0;
    rev:non-backwards-compatible;
    description
      "Add new functionality. Remove some deprecated nodes.";
  }

  revision 2019-02-01 {
    rev:label 2.0.0;
    rev:non-backwards-compatible;
    description "Apply bugfix to pattern statement";
  }

  revision 2019-01-01 {
    rev: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:label 2.2.0;
    description "Backwards-compatible bugfix to enhancement.";
  }

  revision 2019-04-01 {
    rev:label 2.1.0;
    description "Apply enhancement to older release train.";
  }

  revision 2019-02-01 {
    rev:label 2.0.0;
    rev:non-backwards-compatible;
    description "Apply bugfix to pattern statement";
  }

  revision 2019-01-01 {
    rev:label 1.0.0;
    description "Initial revision";
  }

  //YANG module definition starts here
}
```

4. Guidance for revision selection on imports

[RFC7950] and [RFC6020] allow YANG module "import" statements to optionally require the imported module to have a specific revision date. In practice, importing a module with an exact revision date can be too restrictive because it requires the importing module to be updated whenever any change to the imported module occurs, and hence section Section 7.1 suggests that authors do not restrict YANG module imports to exact revision dates.

Instead, for conformance purposes (section 5.6 of [RFC7950]), the recommended approach for defining the relationship between specific YANG module revisions is to specify the relationships outside of the YANG modules, e.g., via YANG library [RFC8525], YANG packages [I-D.ietf-netmod-yang-packages], a filesystem directory containing a set of consistent YANG module revisions, or a revision control system commit label.

4.1. Recommending a minimum revision for module imports

Although the previous section indicates that the actual relationship constraints between different revisions of YANG modules should be specified outside of the modules, in some scenarios YANG modules are designed to be loosely coupled, and implementors may wish to select sets of YANG module revisions that are expected to work together. For these cases it can be helpful for a module author to provide guidance on a recommended minimum revision that is expected to satisfy an YANG import. E.g., the module author may know of a dependency on a type or grouping that has been introduced in a particular imported YANG module revision. Although there can be no guarantee that all derived future revisions from the particular imported module will necessarily also be compatible, older revisions of the particular imported module are very unlikely to ever be compatible.

This document introduces a new YANG extension statement to provide guidance to module implementors on a recommended minimum module revision of an imported module that is anticipated to be compatible. This statement has been designed to be machine-readable so that tools can parse the minimum revision extension statement and generate warnings if appropriate, but this extension statement does not alter YANG module conformance of valid YANG module versions in any way, and specifically it does not alter the behavior of the YANG module import statement from that specified in [RFC7950].

The ietf-revisions module defines the "recommended-min" extension statement, a substatement to the YANG "import" statement, to allow for a "minimum recommended revision" to be documented:

> The argument to the "recommended-min" extension statement is a revision date or a revision label.

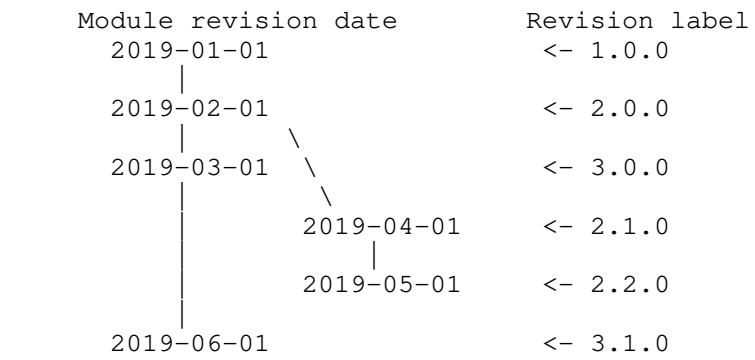
A particular revision of an imported module adheres to an import's "recommended-min" extension statement if the imported module's revision history contains a revision statement with a matching revision date or revision label. Removing entries from a module's revision history may cause a particular revision to no longer satisfy an import's "recommended-min" statement if the revision-date or label is no longer present in the module's revision history; further described in Section 3.3 and Section 7.1.

The "recommended-min" extension statement MAY be specified multiple times, allowing a set of recommended minimum revisions to be documented. Module implementors are recommended to pick a module revision that adheres to any of the "recommended-min" statements.

Adding, modifying or removing a "recommended-min" extension statement is a BC change.

4.1.1. 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:



4.1.1.1. Example 1

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

```
import example-module {
  rev:recommended-min 2019-02-01;
}
```

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

```
import example-module {
  rev:recommended-min 2.0.0;
}
```

4.1.1.2. Example 2

This example recommends module revisions for import 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 recommended revision for import.

```
import example-module {
  rev:recommended-min 2.1.0;
}
```

4.1.1.3. Example 3

This example recommends module revisions for import that are derived from either 2019-04-01 or 2019-06-01. It includes revisions/labels: 2019-04-01/2.1.0, 2019-05-01/2.2.0, and 2019-06-01/3.1.0.

```
import example-module {
  rev:recommended-min 2019-04-01;
  rev:recommended-min 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 [RFC9195] 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:recommended-min" extension MAY 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 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.

```
<CODE BEGINS> file "ietf-yang-revisions@2022-11-29.yang"
module ietf-yang-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-revisions";
  prefix rev;

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

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

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

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

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

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

     Copyright (c) 2002 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; 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 2022-11-29 {
    rev:label "1.0.0-draft-ietf-netmod-yang-module-versioning-08";
    description
      "Initial version.";
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }

  typedef revision-date {
    type string {
      pattern '[0-9]{4}-(1[0-2]|0[1-9])-(0[1-9]|[1-2][0-9]|3[0-1])';
    }
    description
      "A date associated with a YANG revision.

       Matches dates formatted as YYYY-MM-DD.";
    reference
      "RFC 7950: The YANG 1.1 Data Modeling Language";
  }

  typedef revision-label {
    type string {
      length "1..255";
      pattern '[a-zA-Z0-9,\-_.+]+';
      pattern '[0-9]{4}-[0-9]{2}-[0-9]{2}' {
        modifier "invert-match";
        error-message
          "The revision-label must not match a revision-date.";
      }
    }
    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 or pattern similar to a date.";
    reference
      "XXXX: Updated YANG Module Revision Handling;
       Section 3.3, Revision label";
  }

  typedef revision-date-or-label {
    type union {
```

```
        type revision-date;
        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 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 recommended-min {
      argument revision-date-or-label;
      description
        "Recommends 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 'recommended-min' 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 'recommended-min' statements is
         an acceptable recommended revision for import.

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

         Adding, removing or updating a 'recommended-min' statement to
         an import is a backwards-compatible change.";
      reference
        "XXXX: Updated YANG Module Revision Handling; Section 4,
         Recommending a minimum revision for module imports";
    }

    identity revision-label-scheme-base {
      description
        "Base identity from which all revision label schemes are
         derived.";
      reference
        "XXXX: Updated YANG Module Revision Handling;
         Section 3.3.1, Revision label scheme extension statement";
    }
  }
  <CODE ENDS>
```

YANG module with augmentations to YANG Library to revision labels

```
<CODE BEGINS> file "ietf-yang-library-revisions@2021-11-04.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
      "XXXX: Updated YANG Module Revision Handling";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 8525: YANG Library";
  }

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

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

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

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

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

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

     Copyright (c) 2002 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; 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 label version with 1.0.0 and
    // remove this note.

    revision 2021-11-04 {
      rev: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 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 revision label value in the
            specific revision of the submodule included by the module
            loaded in this module-set.";
         reference
           "XXXX: Updated YANG Module Revision Handling;
            Section 5.2.1, Advertising revision-label";
       }
     }

     augment "/yanglib:yang-library/yanglib:module-set/"
           + "yanglib:import-only-module" {
       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 revision label value in the
            specific revision of the module included in this
            module-set.";
         reference
           "XXXX: Updated YANG Module Revision Handling;
            Section 5.2.1, Advertising revision-label";
       }
     }

     augment "/yanglib:yang-library/yanglib:module-set/"
           + "yanglib:import-only-module/yanglib:submodule" {
       description
         "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:label value in the specific
            revision of the submodule included by the import-only-module
            loaded in this module-set.";
         reference
           "XXXX: Updated YANG Module Revision Handling;
            Section 5.2.1, Advertising revision-label";
       }
```

```
    }

    augment "/yanglib:yang-library/yanglib:schema" {
      description
        "Augmentations to the ietf-yang-library module to indicate how
         deprecated and obsoleted nodes are handled for each datastore
         schema supported by the server.";
      leaf deprecated-nodes-implemented {
        type boolean;
        description
          "If set to true, this leaf indicates that all schema nodes
           with a status 'deprecated' 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";
      }
    }
  }
  <CODE ENDS>
```

9. Security considerations

9.1. Security considerations for module revisions

As discussed in the introduction of this document, YANG modules occasionally undergo changes that are not backwards compatible. This occurs in both standards and vendor YANG modules despite the prohibitions in RFC 7950. RFC 7950 also allows nodes to change to status 'obsolete' which can change behavior and compatibility for a client.

The fact that YANG modules change in a non-backwards-compatible manner may have security implications. Such changes should be carefully considered, including the scenarios described below. The rev:non-backwards-compatible extension statement introduced in this document provides an alert that the module or submodule may contain changes that impact users and need to be examined more closely for both compatibility and potential security implications. Flagging the change reduces the risk of introducing silent exploitable vulnerabilities.

When a module undergoes a non-backwards-compatible change, a server may implement different semantics for a given leaf than a client using an older version of the module is expecting. If the particular leaf controls any security functions of the device, or is related to parts of the configuration or state that are sensitive from a security point of view, then the difference in behavior between the old and new revisions needs to be considered carefully. In particular, changes to the default of the leaf should be examined.

Implementors and users should also consider impact to data node access control rules (e.g. The Network Configuration Access Control Model (NACM) [RFC8341]) in the face of non-backwards-compatible changes. Access rules may need to be adjusted when a new module revision is introduced that contains a non-backwards-compatible change.

If the changes to a module or submodule have security implications, it is recommended to highlight those implications in the description of the revision statement.

9.2. Security considerations for the modules defined in this document

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

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

This document does not define any new protocol or data nodes that are writable.

This document updates YANG Library [RFC8525] with augmentations to include revision labels in the YANG library data and two boolean leafs to indicate whether status deprecated and status obsolete schema nodes are implemented by the server. These read-only augmentations do not add any new security considerations beyond those already present in [RFC8525].

10. IANA Considerations

10.1. YANG Module Registrations

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

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

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

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

The ietf-yang-revisions module:

Name: ietf-yang-revisions

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

Prefix: rev

Reference: [RFCXXXX]

The ietf-yang-library-revisions module:

Name: ietf-yang-library-revisions

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

Prefix: yl-rev

Reference: [RFCXXXX]

10.2. 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.

11. References

11.1. Normative References

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-11, 10 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-11>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

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

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

11.2. Informative References

[AddrFamilyReg]
"Address Family Numbers IANA Registry", <https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>.

[I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>.

[I-D.ietf-netmod-yang-packages]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-

ietf-netmod-yang-packages-03, 4 March 2022,
<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-03>.

[I-D.ietf-netmod-yang-schema-comparison]
Andersson, P. and R. Wilton, "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-02, 14 March 2023,
<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-schema-comparison-02>.

[I-D.ietf-netmod-yang-ver-selection]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Schema Selection", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-ver-selection-00, 17 March 2020,
<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-ver-selection-00>.

[I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-versioning-reqs-07, 10 July 2022,
<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-versioning-reqs-07>.

[IfTypesReg]
"Interface Types (ifType) IANA Registry",
<https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,
<https://www.rfc-editor.org/info/rfc8340>.

[RFC9195] Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", RFC 9195, DOI 10.17487/RFC9195, February 2022, <https://www.rfc-editor.org/info/rfc9195>.

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

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

[SAFIReg] "Subsequent Address Family Identifiers (SAFI) Parameters IANA Registry", <https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

Appendix A. Examples of changes that are NBC

Examples of NBC changes include:

* 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 remove any previously allowed values from 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.

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. One possible option is to have the server prevent the new node from being set if the old node is already set (and vice-versa). The new node could have a "when" statement added to it to achieve this. The old node, however, must not have a "when" statement added, or an existing "when" modified to be more restrictive, since this would be an NBC change. In any case, the server could reject the old node from being set if the new node is already set.

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. One possible option is to have the server prevent entries in the new list from being created if the old list already has entries (and vice-versa).

4. When list "sessions" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

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. One possible option is to have the server prevent the new node from being set if the old node is already set (and vice-versa). The new node could have a "when" statement added to it to achieve this. The old node, however, must not have a "when" statement added, or an existing "when" modified to be more restrictive, since this would be an NBC change. In any case, the server could reject the old node from being set if the new node is already set.

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

Contributors

The following people made substantial contributions to this document:

Bo Wu
lana.wubo@huawei.com

Jan Lindblad
jlindbla@cisco.com

Acknowledgments

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

Benoit Claise
benoit.claise@huawei.com

Ebben Aries
exa@juniper.net

Juergen Schoenwaelder
j.shoenwaelder@jacobs-university.de

Mahesh Jethanandani
mjethanandani@gmail.com

Michael (Wangzitao)
wangzitao@huawei.com

Per Andersson
perander@cisco.com

Qin Wu
bill.wu@huawei.com

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, and Kent Watsen for their contributions and review comments.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.
Email: rwilton@cisco.com

Reshad Rahman (editor)
Graphiant
Email: reshad@yahoo.com

Balazs Lengyel (editor)
Ericsson
Email: balazs.lengyel@ericsson.com

Joe Clarke
Cisco Systems, Inc.
Email: jclarke@cisco.com

Jason Sterne
Nokia
Email: jason.sterne@nokia.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: 12 September 2023

P. Andersson, Ed.
R. Wilton
Cisco Systems, Inc.

11 March 2023

YANG Schema Comparison
draft-ietf-netmod-yang-schema-comparison-02

Abstract

This document specifies an algorithm for comparing two revisions of a YANG schema to determine the scope of changes, and a list of changes, between the revisions. The output of the algorithm can be used to help select an appropriate revision-label or YANG semantic version number for a new revision. This document defines a YANG extension that provides YANG annotations to help the tool accurately determine the scope of changes between two revisions.

Table of Contents

1. Key Issues

{ This section is only to present the current ongoing work, not part of the final draft. }

The contributors have identified several key issues that need attention. This section presents selected key issues which have been discussed together with suggestions for proposed solution or requirements.

1.1. On-wire vs Schema analysis

Should one algorithm be used or two? The consesus reached was to define two separate algorithms, one for on-wire format and one for schema.

On the wire: the focus is on what types of changes affect the client requests and server responses for YANG driven protocols, e.g. NETCONF, RESTCONF, gNMI. If the same requests and responses occur, then there is no "on the wire" impact of the change. For example, changing the name of a "choice" has no impact "on the wire". For many clients, this level of compatiblity is enough.

Schema: any changes that affect the YANG schema in an NBC manner according to the full rules of [I-D.ietf-netmod-yang-module-versioning]. This may be important for clients that, for example, automatically generate code using the YANG and where the change of a typedef name or a choice name could be significant. Also important for other modules that may augment or deviate the schema being compared.

Changes to the module that aren’t semantic should raise that there has been editorial changes

Ordering in the schema, RFC 7950 doesn’t allow reordering; thus an NBC change.

Open Questions:

Groupings / uses

typedefs, namespaces, choice names, prefixes, module metadata.

* typedef renaming (on-wire, same base type etc)

* Should all editorial (text) diffs be reported?

* What about editorial changes that might change semantics, e.g. a description of a leaf?

* Metadata arguments which relies on the formatted input text. E.g description, contact (etc), extension (how does the user want to tune verbosity level for editorial changes: whitespace, spelling, editorial, potentially-nbc?

* XPath, must, when: don't normalize XPath expressions

* presence statements

1.2. error-tags, error messages, and other error statements

Error tags and messages might be relied on verbatim by users.

* error-tag: standardized in [RFC6241]

* error-app-tag: arbitrary text ([RFC6241] but also model)

* error-message: arbitrary

Failed must statement, error-message, assumed NBC

Default behaviour is changes to error tags, messages etc are NBC.

1.3. Comparison on module or full schema (YANG artifact, arbitrary blob. Questions

* features

* packages vs directories vs libraries vs artifact

* package specific comparison, package metadata or only looking at the modules

* import only or implemented module

Filter out comparison for a specific subrtree, path etc. Use case for on-wire e.g. yang subscriptions, did the model change fro what is subscribed on?

2. Open Issues

{ This section is only to present the current ongoing work, not part of the final draft. }

The following issues have not ben discussed in any wider extent yet.

2.1. Override/per-node tags

2.2. Separate rules for config vs state

2.3. Tool/report verbosity

* where to report changes (module, grouping, typedef, uses)

* output level (conceptual level or exact strings)

* granularity: error/warning/info level per reported change category

2.4. sub-modules

2.5. Write algorithm in pseudo code or just describe the rules/goals in text?

2.6. Categories in the report: bc, nbc, potentially-nbc, editorial. Allow filtering in the draft without defining it?

One option can be to have a tool option that presents the reason behind the decision, e.g. --details could be used to explain to the user why a certain change was marked as nbc.

Another option is to present reasoning and analysis in deeper levels of verbosity; e.g. one extra level of verbosity, -v, could present the reason for categorizing a change nbc, and an additional extra level of verbosity, e.g. -vv, could also present the detailed analysis the tool made to categorize the change.

2.7. Only for YANG 1.1?

2.8. renamed-from

3. Tool options

{ This section is only to present the current ongoing work, not part of the final draft. }

During the work a list of useful tool options are identified for later discussion and publication in an appendix.

* An option for how to interpret description changes (for the on-wire algorithm) by default, e.g. treat them as editorial or nbc.

* Option: --skip-error-tags, etc

4. Introduction

Warning, this is an early (-00) draft with the intention of scoping the outline of the solution, hopefully for the WG to back the direction of the solution. Refinement of the solution details is expected, if this approach is accepted by the WG.

This document defines a solution to Requirement 2.2 in [I-D.ietf-netmod-yang-versioning-reqs]. Complementary documents provide a complete solution to the YANG versioning requirements, with the overall relationship of the solution drafts described in [I-D.ietf-netmod-yang-solutions].

YANG module 'revision-labels' [I-D.ietf-netmod-yang-module-versioning] and the use of YANG semantic version numbers [I-D.ietf-netmod-yang-semver] can be used to help manage and report changes between revisions of individual YANG modules.

YANG packages [I-D.ietf-netmod-yang-packages] along with YANG semantic version numbers can be used to help manage and report changes between revisions of YANG schema.

[I-D.ietf-netmod-yang-module-versioning] and [I-D.ietf-netmod-yang-packages] define how to classify changes between two module or package revisions, respectively, as backwards compatible or non-backwards-compatible. [I-D.ietf-netmod-yang-semver] refines the definition, to allow backwards compatible changes to be classified as 'minor changes' or 'editorial changes'.

'Revision-label's and YANG semantic version numbers, whilst being generally simple and helpful in the mainline revision history case, are not sufficient in all scenarios. For example, when comparing two revisions/versions on independent revision branches, without a direct ancestor relationship between the two revisions/versions. In this cases, an algorithmic comparison approach is beneficial.

In addition, the module revision history's 'nbc-changes' extension statement, and YANG semantic version numbers, effectively declare the worst case scenario. If any non-backwards-compatible changes are restricted to only parts of the module/schema that are not used by an operator, then the operator is able to upgrade, and effectively treat the differences between the two revisions/versions as backwards compatible because they are not materially impacted by the non-backwards-compatible changes.

Hence, this document defines algorithms that can be applied to revisions of YANG modules or versions of YANG schema (e.g., as represented by YANG packages), to determine the changes, and scope of changes between the revisions/versions.

For many YANG statements, programmatic tooling can determine whether the changes between the statements constitutes a backwards-compatible or non-backwards-compatible change. However, for some statements, it is not feasible for current tooling to determine whether the changes are backwards-compatible or not. For example, in the general case, tooling cannot determine whether the change in a YANG description statement causes a change in the semantics of a YANG data node. If the change is to fix a typo or spelling mistake then the change can be classified as an editorial backwards-compatible change. Conversely, if the change modifies the behavioral specification of the data node then the change would need to be classified as either a non editorial backwards-compatible change or a non-backwards-compatible change. Hence, extension statements are defined to annotate a YANG module with additional information to clarify the scope of changes in cases that cannot be determined by algorithmic comparison.

Open issues are tracked at https://github.com/netmod-wg/yang-ver-dt/issues, tagged with 'schema-comparison'.

5. Terminology and Conventions

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

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

* schema node

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

This document makes of the following terminology introduced in the YANG Packages [I-D.ietf-netmod-yang-packages]:

* YANG schema

In addition, this document defines the terminology:

* Change scope: Whether a change between two revisions is classified as non-backwards-compatible, backwards-compatible, or editorial.

* Node compatibility statement: An extension statements (e.g. nbc-change-at) that can be used to indicate the backwards compatibility of individual schema nodes and specific YANG statements.

6. Generic YANG schema tree comparison algorithm

The generic schema comparison algorithm works on any YANG schema. This could be a schema associated with an individual YANG module, or a YANG schema represented by a set of modules, e.g., specified by a YANG package.

The algorithm performs a recursive tree wise comparison of two revisions of a YANG schema, with the following behavior:

The comparison algorithm primarily acts on the parts of the schema defined by unique identifiers.

Each identifier is qualified with the name of the module that defines the identifier.

Identifiers in different namespaces (as defined in 6.2.1 or RFC 7950) are compared separately. E.g., 'features' are compared separately from 'identities'.

Within an identifier namespace, the identifiers are compared between the two schema revisions by qualified identifier name. The 'renamed-from' extension allow for a meaningful comparison where the name of the identifier has changed between revisions. The 'renamed-from' identifier parameter is only used when an identifier in the new schema revision cannot be found in the old schema revision.

YANG extensions, features, identities, typedefs are checked by comparing the properties defined by their YANG sub-statements between the two revisions.

YANG groupings, top-level data definition statements, rpcs, and notifications are checked by comparing the top level properties defined by their direct child YANG sub-statements, and also by recursively checking the data definition statements.

The rules specified in section 3 of [I-D.ietf-netmod-yang-module-versioning] determine whether the changes are backwards-compatible or non-backwards-compatible.

The rules specified in section 3.2 of [I-D.ietf-netmod-yang-packages] determine whether backwards-compatible changes are 'minor' or 'editorial'.

For YANG "description", "must", and "when" statements, the "backwards-compatible" and "editorial" extension statements can be used to mark instances when the statements have changed in a backwards-compatible or editorial way. Since by default the comparison algorithm assumes that any changes in these statements are non-backwards-compatible. XXX, more info required here, since the revisions in the module history probably need to be available for this to work in the general branched revisions case.

Submodules are not relevant for schema comparison purposes, i.e. the comparison is performed after submodule resolution has been completed.

6.1. YANG module revision scope extension annotations

6.2. Node compatibility extension statements

In addition to the revision extension statement in [I-D.ietf-netmod-yang-module-versioning], this document defines YANG extension statements to indicate compatibility information for individual schema nodes and certain YANG statements.

The node compatibility extension statements are applicable to schema nodes (e.g. leaf, rpc, choice) as defined in [RFC7950], as well as a set of YANG statements (e.g. typedef) as listed in the YANG definition of the nbc-change-at extension in the ietf-yang-revisions module in this document.

While the top level non-backwards-compatible-revision statement is mandatory when there is a non-backwards-compatible change, the node compatibility statements are optional.

For many YANG statements, programmatic tooling can determine whether the changes to a statement between two module revisions constitutes a backwards-compatible or non-backwards-compatible change. However, for some statements, it may be impractical for tooling to determine whether the changes are backwards-compatible or not. For example, in the general case, tooling cannot determine whether the change in a YANG description statement causes a change in the semantics of a YANG schema node. If the change is to fix a typo or spelling mistake then the change can be classified as an editorial backwards-compatible change. Conversely, if the change modifies the behavioral specification of the data node then the change would need to be

classified as either a non editorial backwards-compatible change or a non-backwards-compatible change. Hence, extension statements are defined to annotate a YANG module with additional information to clarify the scope of changes in cases that cannot be determined by algorithmic comparison.

Three extensions are defined for schema node compatibility information:

nbc-change-at: Indicates a specific YANG statement had a non-backwards-compatible change at a particular module or sub-module revision

bc-change-at: Indicates a specific YANG statement had a backwards-compatible change at a particular module or sub-module revision

editorial-change-at: Indicates a specific YANG statement had an editorial change at a particular module or sub-module revision. The meaning of an editorial change is as per YANG Semver [I-D.ietf-netmod-yang-semver]

When a node compatibility statement is added to a schema node in a sub-module, the revision indicated for the compatibility statement is that of the sub-module.

Adding, modifying or removing any of the node compatibility statements is considered to be a BC change.

The following example illustrates the node compatibility statements:

```
                container some-stuff {
                  leaf used-to-be-a-string {
                    rev:nbc-change-at "3.0.0" {
                      description "Changed from a string to a uint32.";
                    }
                    type uint32;
                  }
                  leaf fixed-my-description-typo {
                    rev:editorial-change-at "2022-06-03";
                    type string;
                    description "This description used to have a typo."
                  }
                  list sir-changed-a-lot {
                    rev:editorial-change-at "3.0.0";
                    rev:bc-change-at "2.3.0";
                    rev:bc-change-at "1.2.1_non_compatible";
                    description "a list of stuff";
                    ordered-by user;
                    key "foo";
                    leaf foo {
                      type string;
                    }
                    leaf thing {
                      type uint8;
                    }
                  }
```

Note that an individual YANG statement may have a backwards-compatible change in a revision that is non-backwards-compatible (e.g. some other node changed in a non-backwards-compatible fashion in that particular revision).

If changes are ported from one branch of YANG model revisions to another branch, care must be taken with any node compatibilty statements. A simple copy-n-paste should not be used. The node compatibilty statements may incorrectly reference a revision that is not in the history of the new revision. Further, the statements might not apply depending on what the history is like in that new branch (e.g., an NBC change that is ported might not be an NBC change in the new branch). Node compatiblity statements should not be copied over to the new branch. Instead, the changes should be considered as completely new on the new branch, and any compatibility information should be generated from scratch.

When a node compatibility statement is present, that compatibilty statement is the authoritative classification of the backwards compatibility of the change to the schema node in the specifed revision. This allows a human author to explicitly communicate the compatibilty and potentially override the rules specified in this document. This is useful in a number of situations including:

* When a tool may not be able to accurately determine the compatibilty of a change. For example, a change in a 'pattern' or 'must' statement can be difficult for a user or tool to determine if it is a compatible change.

* When a pattern, range or other statement is changed to more correctly define the server constraint. An example is correcting a pattern that incorrectly included 355.xxx.xxx.xxx as a possible IPv4 address to make it only accept up to 255.xxx.xxx.xxx.

Nothing about the backwards compatibility of a schema node is implied by the absence of a node compatibility statement. Hence, the schema node definition must be compared between the two revisions to determine the backwards compatibility.

If any nbc-change-at extension statements exists in a module or sub-module, then the module or sub-module MUST have non-backwards-compatible-revision substatements in each revision statement of the module or sub-module history where the revision matches the argument of any nbc-change-at statements. If any revision statements are removed, then all node compatibiilty statements that reference that revision MUST also be removed. Conversely, node compatibilty statements MUST NOT be removed unless the associated revision statement in the revision history is removed.

If a node compatiblity statement is added to a grouping, then all instances where the grouping is used in the module or by an importing module are also impacted by the compatibilty information. Similarly for a 'typedef', all leafs and leaf-lists that use that typedef share the specified compatibility classification. A non-backwards-compatible change to a typedef or grouping defined in one module that is used by an importing module, does not cause the importing module to add a non-backwards-compatible-revision statement to the revision history. Non-backwards-compatible marking does not carry through import statements.

A node compatibility statement at a leaf, leaf-list, or typedef context takes precedence over a node compatibility statement in a typedef used by the leaf, leaf-list, or typedef. If multiple typedefs with compatibility statements are used by a leaf, leaf-list, or typedef (e.g. a union), and there is no compatibility statement at

the top leaf, leaf-list, or typedef context, then the order of precedence used to classify the compatibility of the top level leaf, leaf-list, or typedef is as follows: nbc-change-at, bc-change-at, and finally editorial-change-at. That is, the leaf, leaf-list, or typedef takes the most impactful change classification of all the underlying typedefs.

Node compatibility statements are not supported on YANG statements such as 'pattern' or 'range'. The compatibility statement instead goes against the leaf, leaf-list, or typedef context.

Node compatibility statements that refer to pre-release revisions of a module MUST be removed when a full release revision of the module is published.

Node compatibilty statements SHOULD NOT be used when it isn't clear which change the statement is referring to. For example: If a leaf is reordered within a container, a node compatibility statement SHOULD NOT be used against the parent container nor against the reordered leaf. Similarly, if a leaf is renamed or moved to another context without keeping the old leaf present in the model and marked obsolete, a node compatibilty statement SHOULD not be used.

7. YANG module comparison algorithm

The schema comparison algorithm defined in Section 6 can be used to compare the schema for individual modules, but with the following modifications:

Changes to the module's metadata information (i.e. module level description, contact, organization, reference) should be checked (as potential editorial changes).

The module's revision history should be ignored from the comparison.

Changes to augmentations and deviations should be sorted by path and compared.

8. YANG schema comparison algorithms

8.1. Standard YANG schema comparison algorithm

The standard method for comparing two YANG schema versions is to individually compare the module revisions for each module implemented by the schema using the algorithm defined in Section 7 and then aggregating the results together:

* If all implemented modules in the schema have only changed in an editorial way then the schema is changed in an editorial way

* If all implemented modules in the schema have only been changed in an editorial or backwards-compatible way then the schema is changed in a backwards-compatible way

* Otherwise if any implemented module in the schema has been changed in a non-backwards-compatible way then the schema is changed in a non-backwards-compatible way.

The standard schema comparison method is the RECOMMENDED scheme to calculate the version number change for new versions of YANG packages, because it allows the package version to be calculated based on changes to implemented modules revision history (or YANG semantic version number if used to identify module revisions).

8.2. Filtered YANG schema comparison algorithm

Another method to compare YANG schema, that is less likely to report inconsequential differences, is to construct full schema trees for the two schema versions, directly apply a version of the comparison algorithm defined in Section 6. This may be particular useful when the schema represents a complete datastore schema for a server because it allows various filtered to the comparison algorithm to provide a more specific answer about what changes may impact a particular client.

The full schema tree can easily be constructed from a YANG package definition, or alternative YANG schema definition.

Controlled by input parameters to the comparison algorithm, the following parts of the schema trees can optionally be filtered during the comparison:

All "grouping" statements can be ignored (after all "use" statements have been processed when constructing the schema).

All module and submodule metadata information (i.e. module level description, contact, organization, reference) can be ignored.

The comparison can be restricted to the set of features that are of interest (different sets of features may apply to each schema versions).

The comparison can be restricted to the subset of data nodes, RPCs, notifications and actions, that are of interest (e.g., the subset actually used by a particular client), providing a more meaningful result.

The comparison could filter out backwards-compatible 'editorial' changes.

In addition to reporting the overall scope of changes at the schema level, the algorithm output can also optionally generate a list of specific changes between the two schema, along with the classification of those individual changes.

9. Comparison tooling

'pyang' has some support for comparison two module revisions, but this is currently limited to a linear module history.

TODO, it would be helpful if there is reference tooling for schema comparison.

10. Module Versioning Extension YANG Modules

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

<CODE BEGINS> file "ietf-yang-rev-annotations@2023-02-14.yang"

```
module ietf-yang-rev-annotations {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-rev-annotations";
  prefix rev-ext;

  import ietf-yang-revisions {
    prefix rev;
  }

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

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

  description
    "This YANG 1.1 module contains extensions to annotation to YANG
     module with additional metadata information on the nature of
```

```
        changes between two YANG module revisions.

        XXX, maybe these annotations could also be included in
        ietf-yang-revisions?

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

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

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

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

    revision 2023-03-11 {
      rev:revision-label 1.0.0-draft-ietf-netmod-yang-schema-comparison-02;
      description
        "Draft revision";
      reference
        "XXXX: YANG Schema Comparison";
    }

    extension nbc-change-at {
      argument revision-date-or-label;
      description
        "A node compatibility statement that identifies a revision
        (by revision-label, or revision date if a revision-label is
        not available) where a non-backwards-compatible change has
        occurred in a particular YANG statement relative to the
        previous revision listed in the revision history.

        The format of the revision-label argument MUST conform to the
        pattern defined for the ietf-yang-revisions
```

```
        revision-date-or-label typedef.

        The following YANG statements MAY have zero or more
        nbc-change-at substatements:
          - all schema node statements (leaf, rpc, choice, etc)
          - 'feature' statements
          - 'grouping' statements
          - 'identity' statements
          - 'must' statements
          - 'refine' statements
          - 'typedef' statements
          - YANG extensions

        Each YANG statement MUST only a have a single node
        compatibilty statement (one of nbc-change-at, bc-change-at,
        or editorial-change-at) for a particular revision. When a node
        has more than one of the node compatibilty statements (for
        different revisions), they must be ordered from most recent
        to least recent.

        An nbc-change-at statement can have 0 or 1 'description'
        substatements.

        The nbc-change-at statement in not inherited by descendants
        in the schema tree. It only applies to the specific YANG
        statement with which it is associated.
        ";

      reference
        "XXXX: YANG Schema Comparison;
         Section XXX, XXX";

    }

    extension bc-change-at {
      argument revision-date-or-label;
      description
        "A node compatibility statement that identifies a revision
        (by revision-label, or revision date if a revision-label is
        not available) where a backwards-compatible change has
        occurred in a particular YANG statement relative to the
        previous revision listed in the revision history.

        The format of the revision-label argument MUST conform to the
        pattern defined for the ietf-yang-revisions
        revision-date-or-label typedef.

        The following YANG statements MAY have zero or more
```

```
        bc-change-at substatements:
          - all schema node statements (leaf, rpc, choice, etc)
          - 'feature' statements
          - 'grouping' statements
          - 'identity' statements
          - 'must' statements
          - 'refine' statements
          - 'typedef' statements
          - YANG extensions

        Each YANG statement MUST only a have a single node
        compatibilty statement (one of nbc-change-at, bc-change-at,
        or editorial-change-at) for a particular revision. When a node
        has more than one of the node compatibilty statements (for
        different revisions), they must be ordered from most recent
        to least recent.

        An bc-change-at statement can have 0 or 1 'description'
        substatements.

        The bc-change-at statement in not inherited by descendants
        in the schema tree. It only applies to the specific YANG
        statement with which it is associated.
        ";

      reference
        "XXXX: YANG Schema Comparison;
         Section XXX, XXX";

    }

    extension editorial-change-at {
      argument revision-date-or-label;
      description
        "A node compatibility statement that identifies a revision
        (by revision-label, or revision date if a revision-label is
        not available) where an editorial change has
        occurred in a particular YANG statement relative to the
        previous revision listed in the revision history.

        The format of the revision-label argument MUST conform to the
        pattern defined for the ietf-yang-revisions
        revision-date-or-label typedef.

        The following YANG statements MAY have zero or more
        editorial-change-at substatements:
          - all schema node statements (leaf, rpc, choice, etc)
          - 'feature' statements
```

```
            - 'grouping' statements
            - 'identity' statements
            - 'must' statements
            - 'refine' statements
            - 'typedef' statements
            - YANG extensions

          Each YANG statement MUST only a have a single node
          compatibilty statement (one of nbc-change-at, bc-change-at,
          or editorial-change-at) for a particular revision. When a node
          has more than one of the node compatibilty statements (for
          different revisions), they must be ordered from most recent
          to least recent.

          An editorial-change-at statement can have 0 or 1 'description'
          substatements.

          The editorial-change-at statement in not inherited by descendants
          in the schema tree. It only applies to the specific YANG
          statement with which it is associated.
          ";

        reference
          "XXXX: YANG Schema Comparison;
           Section XXX, XXX";

      }

      extension backwards-compatible {
        argument revision-date-or-label;
        description
          "Identifies a revision (by revision-label, or revision date if
           a revision-label is not available) where a
           backwards-compatible change has occurred relative to the
           previous revision listed in the revision history.

           The format of the revision-label argument MUST conform to the
           pattern defined for the ietf-yang-revisions
           revision-date-or-label typedef.

           The following YANG statements MAY have zero or more
           'rev-ext:non-backwards-compatible' statements:
               description
               must
               when

           Each YANG statement MUST only a have a single
           non-backwards-compatible, backwards-compatible, or editorial
```

```
          extension statement for a particular revision-label, or
          corresponding revision-date.";

      reference
        "XXXX: YANG Schema Comparison;
         Section XXX, XXX";
    }

    extension editorial {
      argument revision-date-or-label;
      description
        "Identifies a revision (by revision-label, or revision date if
         a revision-label is not available) where an editorial change
         has occurred relative to the previous revision listed in the
         revision history.

         The format of the revision-label argument MUST conform to the
         pattern defined for the ietf-yang-revisions
         revision-date-or-label typedef.

         The following YANG statements MAY have zero or more
         'rev-ext:non-backwards-compatible' statements:
             description

         Each YANG statement MUST only a have a single
         non-backwards-compatible, backwards-compatible, or editorial
         extension statement for a particular revision-label, or
         corresponding revision-date.";

      reference
        "XXXX: YANG Schema Comparison;
         Section XXX, XXX";
    }

    extension renamed-from {
      argument yang-identifier;
      description
        "Specifies a previous name for this identifier.

         This can be used when comparing schema to optimize handling
         for data nodes that have been renamed rather than naively
         treated them as data nodes that have been deleted and
         recreated.

         The argument 'yang-identifier' MUST take the form of a YANG
         identifier, as defined in section 6.2 of RFC 7950.

         Any YANG statement that takes a YANG identifier as its
```

```
          argument MAY have a single 'rev-ext:renamed-from'
          sub-statement.

          TODO, we should also facilitate identifiers being moved into
          other modules, e.g. by supporting a module-name qualified
          identifier.";

      reference
        "XXXX: YANG Schema Comparison;
         Section XXX, XXX";
    }
  }
  <CODE ENDS>
```

11. 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
* Jason Sterne
* Joe Clarke
* Juergen Schoenwaelder
* Mahesh Jethanandani
* Michael Wang
* Qin Wu
* Reshad Rahman
* Rob Wilton
* Jan Lindblad
* Per Andersson

The ideas for a tooling based comparison of YANG module revisions was first described in [I-D.clacla-netmod-yang-model-update]. This document extends upon those initial ideas.

12. Security Considerations

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

13. IANA Considerations

13.1. YANG Module Registrations

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

The ietf-yang-rev-annotations module:

Name: ietf-yang-rev-annotations

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-rev-annotations

Prefix: rev-ext

Reference: [RFCXXXX]

14. References

14.1. Normative References

[I-D.ietf-netmod-yang-module-versioning]
Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-08, 12 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning-08>.

[I-D.ietf-netmod-yang-packages]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-03, 4 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-03>.

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-10, 17 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-10>.

[I-D.ietf-netmod-yang-solutions]
Wilton, R., "YANG Versioning Solution Overview", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-solutions-01, 2 November 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-solutions-01>.

[I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-versioning-reqs-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-versioning-reqs-07>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

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

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

14.2. Informative References

[I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>.

Authors' Addresses

Per Andersson (editor)
Cisco Systems, Inc.
Email: perander@cisco.com

Robert Wilton
Cisco Systems, Inc.
Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Updates: 8407 (if approved)
Intended status: Standards Track
Expires: 12 October 2023

J. Clarke, Ed.
R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman
Graphiant
B. Lengyel
Ericsson
J. Sterne
Nokia
B. Claise
Huawei

10 April 2023

YANG Semantic Versioning
draft-ietf-netmod-yang-semver-11

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 RFCAAAA-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 12 October 2023.

Copyright Notice

Table of Contents

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 [I-D.ietf-netmod-yang-module-versioning] defines a revision-label which can be used as an alias to provide additional context or as a meaningful label to refer to a specific revision.

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.

* SemVer: A version string that corresponds to the rules defined in [SemVer] . This specific camel-case notation is the one used by the SemVer 2.0.0 website and used within this document to distinguish between YANG Semver.

* 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. Relationship Between SemVer and YANG Semver

[SemVer] is completely compatible with YANG Semver in that a SemVer semantic version number is legal according to the YANG Semver rules (though the inverse is not necessarily true). YANG Semver is a superset of the SemVer rules, and allow for limited branching within YANG artifacts. If no branching occurs within a YANG artifact (i.e., you do not use the compatibility modifiers described below), the YANG Semver version label will appear as a SemVer version number.

3.2. 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 YANG 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.

3.3. 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 lifecyles 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 "_compatible" or "_non_compatible" modifier.

* 'Z' is the PATCH version. Changes in the PATCH version number can indicate an editorial change to the YANG artifact. In conjunction with the '_COMPAT' modifier (see below) changes to 'Z' may indicate a more substantive module 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.

* '_COMPAT' is an additional modifier, unique to YANG Semver (i.e., not valid in [SemVer]), that indicates 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.

- 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

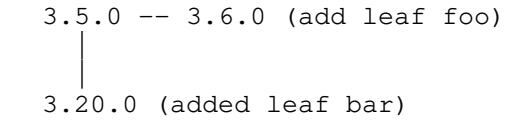
The '_COMPAT' modifier string is "sticky". Once a revision of a module has a modifier in the revision label, then all descendants of that revision with the same X.Y version digits will also have a modifier. The modifier can change from "_compatible" to "_non_compatible" in a descendant revision, but 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.

3.3.1. Branching Limitations with YANG Semver

YANG artifacts that use the YANG Semver revision-label scheme MUST ensure that two artifacts with the same MAJOR version number and no _compatible or _non_compatible modifiers are backwards compatible. Therefore, certain branching schemes cannot be used with YANG Semver. For example, the following branched parent-child module relationship using the following YANG Semver revision labels is not supported:



In this case, given only the revision labels 3.6.0 and 3.20.0 without any parent-child relationship information, one would assume that 3.20.0 is backwards compatible with 3.6.0. But in the illegal example above, 3.20.0 is not backwards compatible with 3.6.0 since 3.20.0 does not contain the leaf foo.

Note that this type of branched parent-child relationship, where two revisions have different backwards compatible changes based on the same parent, is allowed in [I-D.ietf-netmod-yang-module-versioning] .

3.3.2. 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 3.2 and Section 3.3 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

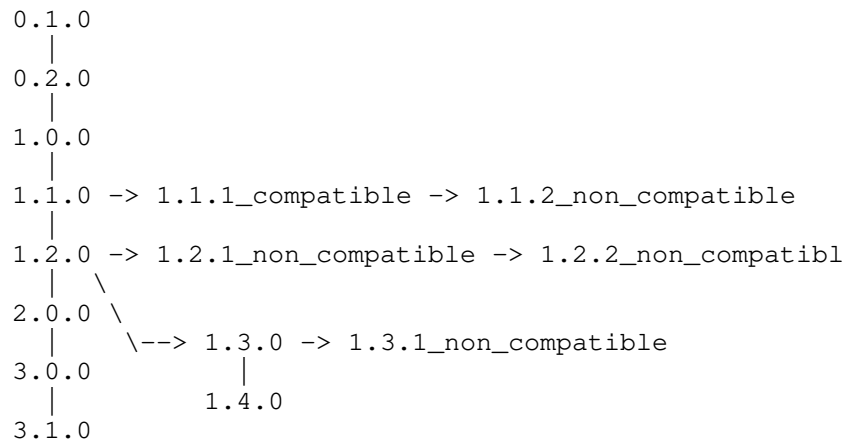
3. The submodule with the schema definitions added has backwards-compatible changes

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).

3.3.3. 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".

3.4. 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.

3.5. Examples of the YANG Semver Label

3.5.1. 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: YANG 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;
```

```
      //   rev:non-backwards-compatible; // optional
      //                           // (theoretically no
      //                           // 'previous released version')
      // }

      // revision 2017-01-26 {
      //   description "Initial module version";
      //   rev:revision-label 0.1.0;
      // }

      //YANG module definition starts here
    }
```

3.5.2. Example of Package Using YANG Semver

Below is an example YANG package that uses the YANG Semver revision label based on the rules defined in this document. Note: '\' line wrapping per [RFC8792] .

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-yang-pkg",
    "content-schema": {
      "module": "ietf-yang-packages@2022-03-04"
    },
    "timestamp": "2022-12-06T17:00:38Z",
    "description":  ["Example of a Package  \
       using YANG Semver"],
    "content-data": {
      "ietf-yang-packages:packages": {
        "package": [
          {
            "name": "example-yang-pkg",
            "version": "1.3.1",
            ...
          }
        ]
      }
    }
  }
}
```

Figure 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:

```
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.

5.2.1. Guidelines for IETF Module Development

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.

Some draft revisions may not include an update to the YANG modules or submodules contained in the draft. In that case, those modules or submodules that are not updated do not not require a change to their versions. Updates to the YANG Semver version MUST only be done when the revision of the module changes.

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

5.2.2. Guidelines for Published IETF Modules

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 3.4 . 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).

6. YANG Module

This YANG module contains the typedef for the YANG semantic version and the identity to signal its use.

```
<CODE BEGINS> file "ietf-yang-semver@2023-01-17.yang"
module ietf-yang-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-semver";
  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) 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
       (http://trustee.ietf.org/license-info).

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

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.
    // RFC Ed. update the rev:revision-label to "1.0.0".

    revision 2023-01-17 {
      rev:label "1.0.0-draft-ietf-netmod-yang-semver-10";
      description
        "Initial revision";
      reference
        "RFC XXXX: YANG Semantic Versioning.";
    }

    /*
     * Identities
     */

    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.";
    }

    /*
     * Typedefs
     */

    typedef version {
      type rev:revision-label {
        pattern '[0-9]+[.][0-9]+[.][0-9]+(_(non_)?compatible)?'
              + '(-[A-Za-z0-9.-]+[.-][0-9]+)?([+][A-Za-z0-9.-]+)?';
      }
      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.";
    }
  }
  <CODE ENDS>
```

7. Contributors

The following people made substantial contributions to this document:

Bo Wu
lana.wubo@huawei.com

Jan Lindblad
jlindbla@cisco.com

Figure 2

8. Acknowledgments

This document grew out of the YANG module versioning design team that started after IETF 101. The 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), Per Andersson, Qin Wu, Reshad Rahman, Tom Hill, and Rob Wilton.

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

We would also like to thank Joseph Donahue from the SemVer.org project for his input on SemVer use and overall document readability.

9. Security Considerations

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

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

That said, the YANG module in this document does not define any schema nodes (i.e., nothing that can be read or written). It only defines a typedef and an identity. Therefore, there is no need to further protect any nodes with access control.

10. IANA Considerations

10.1. 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.

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

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-semver module:

Name: ietf-yang-semver

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

Prefix: ysver

Reference: [RFCXXXX]

10.2. 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 3.4 .

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 3.4 .

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.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

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

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[I-D.ietf-netmod-yang-module-versioning] Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-08, 12 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning-08>.

11.2. Informative References

[I-D.clacla-netmod-yang-model-update] Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>.

[I-D.ietf-netmod-yang-packages]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-03, 4 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-03>.

[I-D.ietf-netmod-yang-schema-comparison]
Andersson, P. and R. Wilton, "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-02, 14 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-schema-comparison-02>.

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[openconfigsemver]
"Semantic Versioning for Openconfig Models",
<http://www.openconfig.net/docs/semver/>.

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

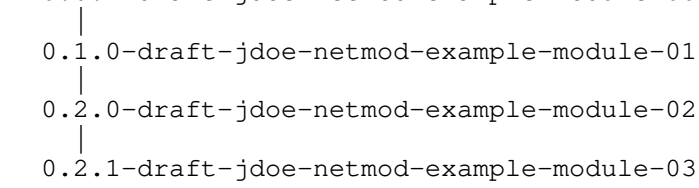
[IfTypeYang]
"iana-if-type YANG Module",
<https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>.

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

Appendix A. Example IETF Module Development

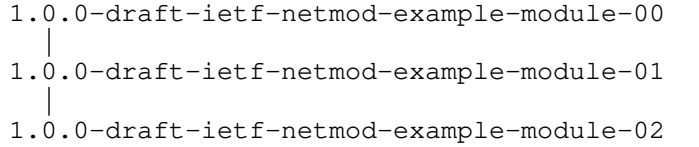
Assume a new YANG module is being developed in the netmod working group in the IETF. Initially, this module is being developed in an individual internet draft, draft-jdoe-netmod-example-module. The following represents the initial version tree (i.e., value of revision-label) of the module as it's being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

Version lineage for initial module development:



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:



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-asmith-netmod-exmod-changes. These are initially developed in parallel with the following versions.

Parallel development for next module revision (track 1):

```
1.1.0-draft-jdoe-netmod-exmod-enhancements-00
  |
1.1.0-draft-jdoe-netmod-exmod-enhancements-01
```

In parallel with (track 2):

```
1.1.0-draft-asmith-netmod-exmod-changes-00
  |
1.1.0-draft-asmith-netmod-exmod-changes-01
```

At this point, the WG decides to merge some aspects of both and adopt the work in asmith'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

Robert Wilton (editor)
Cisco Systems, Inc.
Email: rwilton@cisco.com

Reshad Rahman
Graphiant
Email: reshad@yahoo.com

Balazs Lengyel
Ericsson
1117 Budapest
Magyar Tudosok Korutja
Hungary
Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Jason Sterne
Nokia
Email: jason.sterne@nokia.com

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: 10 January 2024

Q. Ma
Q. Wu
Huawei
B. Lengyel
Ericsson
H. Li
HPE

9 July 2023

YANG Extension and Metadata Annotation for Immutable Flag
draft-ma-netmod-immutable-flag-08

Abstract

This document defines a way to formally document existing behavior, implemented by servers in production, on the immutability of some system configuration nodes, using a YANG "extension" and a YANG metadata annotation, both called "immutable", which are collectively used to flag which nodes are immutable.

Clients may use "immutable" statements in the YANG, and annotations provided by the server, to know beforehand when certain otherwise valid configuration requests will cause the server to return an error.

The immutable flag is descriptive, documenting existing behavior, not proscriptive, dictating server behavior.

Table of Contents

1. Introduction

This document defines a way to formally document as a YANG extension or YANG metadata an existing model handling behavior that is already allowed in YANG and has been used by multiple standard organizations and vendors. It is the aim to create one single standard solution for documenting modification restrictions on data declared as configuration, instead of the multiple existing vendor and organization specific solutions. See Appendix B for existing implementations.

YANG [RFC7950] is a data modeling language used to model both state and configuration data, based on the "config" statement. However, there exists some system configuration data that cannot be modified by the client (it is immutable), but still needs to be declared as "config true" to:

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

* place "when", "must" and "leafref" constraints between configuration and immutable data nodes.

* 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;

Client attempts to override an immutable system configuration node are always rejected by the server [I-D.ietf-netmod-system-config]. If the server knows that it will always reject the modification because it internally think it immutable, it should document this towards the clients in a machine-readable way.

This document defines a way to formally document existing behavior, implemented by servers in production, on the immutability of some system configuration nodes, using a YANG "extension" [RFC7950] and a YANG metadata annotation [RFC7952], both called "immutable", which are collectively used to flag which nodes are immutable.

The "immutable" YANG extension is used when the behavior is independent of instances and can be described at the schema-level, while the "immutable" metadata annotation is used when the behavior must be described at the YANG "list" or "leaf-list" instance level.

Comment: Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema?

Immutability is an existing model handling practice. This document does not apply to the server which does not have any immutable system configuration. While in some cases it may be needed, it also has disadvantages, therefore it SHOULD be avoided wherever possible.

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

UC1 Modeling of server capabilities

UC2 HW based auto-configuration

UC3 Predefined Access control Rules

UC4 Declaring immutable system configuration from an LNE's perspective

Appendix A describes the use cases in detail.

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]:

* configuration data

The following terms are defined in [RFC7950]:

* data node

* leaf

* leaf-list

* container

* list
* anydata
* anyxml
* interior node
* data tree

The following terms are defined in [RFC8341]:

* access operation
* write access

The following terms are defined in this document:

immutable flag: A read-only state value the server provides to describe system data it considers immutable. In schema, the immutability of data nodes is conveyed via a YANG "extension" statement. In instance representations, the immutability of data nodes is conveyed via a YANG metadata annotation. Both the extension statement and the metadata annotation are called "immutable". Together, they are alternative ways to express the same behavior.

1.2. Applicability

This document focuses on the configuration which can only be created, updated and deleted by the server, thus cannot be created, updated and deleted by the client.

The "immutable" concept defined in this document only documents existing write access restrictions to writable datastores, given the client is never allowed to edit read-only datastores. The immutable annotation information is also visible even in read-only datastores like <system> (if exists), <intended> and <operational> when a "with-immutable" parameter is carried (see Section 6.2), however this only serves as descriptive information about the instance node itself, but has no effect on the handling of the read-only datastore.

A particular data node or instance has the same immutability in all writable datastores. The immutability of data nodes is protocol and user independent. The immutability and configured value of an existing node must only change by software upgrade or hardware resource/license change.

2. Solution Overview

Immutable configuration can only be created by the system regardless of the implementation of the system configuration datastore [I-D.ietf-netmod-system-config]. If the server implements <system>, immutable configuration is present in <system>. It may be updated or deleted depending on factors like software upgrade or hardware resources/license change. Immutable configuration does not affect the contents of <running> by default.

A client may create/delete immutable nodes with same values as found in <system> (if exists) in read-write configuration datastore (e.g., <running>), which merely mean making immutable nodes visible/ invisible in read-write configuration datastore (e.g., <running>).

If a client tries to override immutable nodes with different values from ones in <system> (if exists), an error is always returned. This document allows the existing immutable system nodes to be formally documented by YANG extension or metadata annotation rather than be written as plain text in the description statement.

Servers reject client's request for updating configuration data when they internally think it immutable. 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.2.1.

Servers adding the immutable property which does not have any additional semantic meaning is discouraged. For example, a key leaf that is given a value and cannot be modified once a list entry is created.

The "immutable" flag is intended to be descriptive.

3. Use of "immutable" Flag for Different Statements

This section defines what the immutable flag means to the client for each YANG data node statement. Whilst this section describes immutability at the schema level, it applies equally to when the immutable flag is set via the metadata annotation on node instances.

Throughout this section, the word "change" refers to create, update, and delete.

3.1. The "leaf" Statement

When a leaf node is immutable, its value cannot change.

3.2. The "leaf-list" Statement

When a leaf-list data node is immutable, its value cannot change.

When the "immutable" YANG extension statement is used on a leaf-list data node, or if a leaf-list inherits immutability from an ancestor, it means that the leaf-list as a whole cannot change: entries cannot be added, removed, or reordered, in case the leaf-list is "ordered-by user".

3.3. The "container" Statement

When a container data node is immutable, its instance cannot change, unless the immutability of its descendant node is toggled.

By default, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

3.4. The "list" Statement

When a list data node is immutable, its instance cannot change, unless the immutability of its descendant node is toggled, per the description elsewhere in this section.

By default, as with all interior nodes, immutability is recursively applied to descendants (see Section 4). This statement is applicable only to the "immutable" YANG extension, as the "list" node does not itself appear in data trees.

3.5. The "anydata" Statement

When an anydata data node is immutable, its instance cannot change. Additionally, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

Descendants for anydata data node is unknown at module design time, they cannot reset the immutability state with "immutable" YANG extension.

3.6. The "anyxml" Statement

When an "anyxml" data node is immutable, its instance cannot change. Additionally, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

Descendants for anyxml data node is unknown at module design time, they cannot reset the immutability state with "immutable" YANG extension.

4. Immutability of Interior Nodes

Immutability is a conceptual operational state value that is recursively applied to descendants, which may reset the immutability state as needed, thereby affecting their descendants. There is no limit to the number of times the immutability state may change in a data tree.

For example, given the following application configuration XML snippets:

```
<application im:immutable="true">
  <name>predefined-ftp</name>
  <protocol>ftp</protocol>
  <port-number im:immutable="false">69</port-number>
</application>
```

The list entry named "predefined-ftp" is immutable="true", but its child node "port-number" has the immutable="false" (thus the client can override this value). The other child node (e.g., "protocol") not specifying its immutability explicitly inherits immutability from its parent node thus is also immutable="true".

5. "Immutable" YANG Extension

5.1. Definition

If servers always reject client modification attempts to some data node that they internally think immutable and irrelevant to its instance data, an "immutable" YANG extension can be used to formally indicate to the clients.

The "immutable" YANG extension can be a substatement to a "config true" leaf, leaf-list, container, list, anydata or anyxml statement. It has no effect if used as a substatement to a "config false" node, but can be allowed anyway.

The "immutable" YANG extension defines an argument statement named "value" which is a boolean type to indicate that whether the node is immutable or not. If the "immutable" YANG extension is not specified for a particular data node, the default immutability is the same as that of its parent node. The immutability for a top-level data node is "false" by default.

6. "Immutable" Metadata Annotation

6.1. Definition

If servers always reject clients modification to some particular instance that they internally think immutable, an "immutable" metadata annotation can be used to formally indicate to the clients.

The "immutable" metadata annotation takes as an value which is a boolean type, it is not returned unless a client explicitly requests through a "with-immutable" parameter (see Section 6.2). If the "immutable" metadata annotation for data node instances is not specified, the default "immutable" value is the same as the immutability of its parent node in the data tree. The immutable metadata annotation value for a top-level instance node is false if not specified.

Note that "immutable" metadata annotation is used to annotate data node instances. A list may have multiple entries/instances in the data tree, "immutable" can annotate some of the instances as read-only, while others are read-write.

6.2. "with-immutable" Parameter

The YANG model defined in this document (see Section 8) augments the <get-config>, <get> operation defined in RFC 6241, and the <get-data> operation defined in RFC 8526 with a new parameter named "with-immutable". When this parameter is present, it requests that the server includes "immutable" metadata annotations in its response.

This parameter may be used for read-only configuration datastores, e.g., <system> (if exists), <intended> and <operational>, but the "immutable" metadata annotation returned indicates the immutability towards read-write configuration datastores, e.g., <startup>, <candidate> and <running>. If the "immutable" metadata annotation for returned child nodes are omitted, it has the same immutability as its parent node. The immutability of top hierarchy of returned nodes is false by default.

Note that "immutable" metadata annotation is not included in a response unless a client explicitly requests them with a "with-immutable" parameter.

7. Interaction between Immutable Flag and NACM

The server rejects an operation request due to immutability when it tries to perform the operation on the request data. It happens after any access control processing, if the Network Configuration Access Control Model (NACM) [RFC8341]is implemented on a server. For example, if an operation requests to override an immutable configuration data, but the server checks the user is not authorized to perform the requested access operation on the request data, the request is rejected with an "access-denied" error.

8. YANG Module

```
<CODE BEGINS>
 file="ietf-immutable@2023-07-09.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;
    }
    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";
    }
    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>

         Author: Hongwei Li
                 <mailto:flycoolman@gmail.com>";

       description
         "This module defines a YANG extension and a metadata annotation
          both called 'immutable', to allow the server to formally
          document existing behavior on the mutability of some
          configuration nodes. Clients may use 'immutable' extension
          statements in the YANG, and annotations provided by the server
          to know beforehand when certain otherwise valid configuration
          requests will cause the server to return an error.

          Copyright (c) 2023 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 2023-05-25 {
         description
           "Initial revision.";
         // RFC Ed.: replace XXXX and remove this comment
         reference
           "RFC XXXX: YANG Extension and Metadata Annotation for
            Immutable Flag";
       }

       extension immutable {
         argument value;
         description
```

```
        "If servers always reject client modification attempts to
         some data node that can only be created, modified and
         deleted by the device itself, an 'immutable' YANG extension
         can be used to formally indicate to the client.

         The statement MUST only be a substatement to a 'config true'
         leaf, leaf-list, container, list, anydata or anyxml
         statement. Zero or one immutable statement per parent
         statement is allowed.

         No substatements are allowed.

         The argument of the 'immutable' statement defines the value,
         indicating whether the node is immutable or not.

         Adding immutable of an existing immutable statement
         is non-backwards compatible changes.
         Other changes to immutable are backwards compatible.";
    }

    md:annotation immutable {
      type boolean;
      description
        "If servers always reject clients modification to some
         particular instance that can only be created, modified and
         deleted by the device itself, an 'immutable' metadata
         annotation can be used to formally indicate to the clients.
         The 'immutable' annotation indicates the immutability of an
         instantiated data node.

         The 'immutable' metadata annotation takes as a value 'true'
         or 'false'. If the 'immutable' metadata annotation for data
         node instances is not specified, the default value is false.
         Explicitly annotating instances as immutable=true has the
         same effect as not specifying this value.";
    }

    grouping with-immutable-grouping {
      description
        "define the with-immutable grouping.";
      leaf with-immutable {
        type empty;
        description
          "If this parameter is present, the server will return the
           'immutable' annotation for configuration that it
           internally thinks it immutable. When present, this
           parameter allows the server to formally document existing
           behavior on the mutability of some configuration nodes.";
```

```
        }
      }
      augment "/ncds:get-data/ncds:input" {
        description
          "Allows the server to include 'immutable' metadata
           annotations in its response to get-data operation.";
        uses with-immutable-grouping;
      }
      augment "/nc:get-config/nc:input" {
        description
          "Allows the server to include 'immutable' metadata
           annotations in its response to get-config operation.";
        uses with-immutable-grouping;
      }
      augment "/nc:get/nc:input" {
        description
          "Allows the server to include 'immutable' metadata
           annotations in its response to get operation.";
        uses with-immutable-grouping;
      }
    }
<CODE ENDS>
```

9. IANA Considerations

9.1. The "IETF XML" Registry

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

```
URI: urn:ietf:params:xml:ns:yang:ietf-immutable
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.
```

9.2. 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
namespace: urn:ietf:params:xml:ns:yang:ietf-immutable
RFC: XXXX
// RFC Ed.: replace XXXX and remove this comment
```

10. Security Considerations

The YANG module specified in this document defines a YANG extension and a metadata Annotation. These can be used to further restrict write access but cannot be used to extend access rights.

This document does not define any protocol-accessible data nodes.

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.

Acknowledgements

Thanks to Kent Watsen, Andy Bierman, Robert Wilton, Jan Lindblad, Reshad Rahman, Anthony Somerset, Lou Berger, Joe Clarke, Scott Mansfield for reviewing, and providing important input to, this document.

References

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

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

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

Informative References

[I-D.ietf-netmod-system-config] Ma, Q., Wu, Q., and C. Feng, "System-defined Configuration", Work in Progress, Internet-Draft, draft-ietf-netmod-system-config-02, 4 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-system-config-02>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC8530] Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Logical Network Elements", RFC 8530, DOI 10.17487/RFC8530, March 2019, <https://www.rfc-editor.org/info/rfc8530>.

[TR-531] ONF, "UML to YANG Mapping Guidelines, <https://wiki.opennetworking.org/download/attachments/376340494/Draft_TR-531_UML-YANG_Mapping_Gdls_v1.1.03.docx?version=5&modificationDate=1675432243513&api=v2>", February 2023.

[TS28.623] 3GPP, "Telecommunication management; Generic Network Resource Model (NRM) Integration Reference Point (IRP); Solution Set (SS) definitions, <https://www.3gpp.org/ftp/Specs/archive/28_series/28.623/28623-i02.zip>".

[TS32.156] 3GPP, "Telecommunication management; Fixed Mobile Convergence (FMC) Model repertoire, <https://www.3gpp.org/ftp/Specs/archive/32_series/32.156/32156-h10.zip>".

Appendix A. Detailed Use Cases

A.1. UC1 - Modeling of server capabilities

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 data 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 making it unchangable. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

A.2. UC2 - HW based auto-configuration - Interface Example

This section shows how to use 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> (if exists, see [I-D.ietf-netmod-system-config]). 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:

```
container interfaces {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf type {
      im:immutable;
      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.2.1. Error Response to Client Updating the Value of an Interface Type

This section shows an example of an error response due to the client modifying an immutable configuration.

Assume the system creates an interface entry named "eth0" given that an inerface is inserted into the device. 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 request will be rejected by the server:

```
<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.3. UC3 - Predefined Access control Rules

Setting up detailed rules for access control is a complex task. (see [RFC8341]) A vendor may provide an initial, predefined set of groups and related access control rules so that the customer can use access control out-of-the-box. The customer may continue using these predefined rules or may add his own groups and rules. The predefined groups shall not be removed or altered guaranteeing that access control remains usable and basic functions e.g., a system-security-administrator are always available.

The system needs to protect the predefined groups and rules, however, the list "groups" or the list "rule-list" cannot be marked as config=false or with the "immutable" extension in the YANG model because that would prevent the customer adding new entries. Still it

would be good to notify the client in a machine readable way that the predefined entries cannot be modified. When the client retrieves access control data the immutable="true" metadata annotation should be used to indicate to the client that the predefined groups and rules cannot be modified.

A.4. UC4 - Declaring immutable system configuration from an LNE's perspective

An LNE (logical network element) is an independently managed virtual network device made up of resources allocated to it from its host or parent network device [RFC8530]. The host device may allocate some resources to an LNE, which from an LNE's perspective is provided by the system and may not be modifiable.

For example, a host may allocate an interface to an LNE with a valid MTU value as its management interface, so that the allocated interface should then be accessible as the LNE-specific instance of the interface model. The assigned MTU value is system-created and immutable from the context of the LNE.

Appendix B. Existing implementations

There are already a number of full or partial implementations of immutability.

3GPP TS 32.156 [TS32.156] and 28.623 [TS28.623]: Requirements and a partial solution

ITU-T using ONF TR-531[TR-531] concept on information model level but no YANG representation.

Ericsson: requirements and solution

YumaPro: requirements and solution

Nokia: partial requirements and solution

Huawei: partial requirements and solution

Cisco using the concept at least in some YANG modules

Junos OS provides a hidden and immutable configuration group called junos-defaults

Appendix C. Changes between revisions

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

v06 - v07

* Use a Boolean type for the immutable value in YANG extension and metadata annotation

* Define a "with-immutable" parameter and state that immutable metadata annotation is not included in a response unless a client explicitly requests them with a "with-immutable" parameter

* reword the abstract and related introduction section to highlight immutable flag is descriptive

* Add a new section to define immutability of interior nodes, and merge with "Inheritance of Immutable configuration" section

* Add a new section to define what the immutable flag means for each YANG data node

* Define the "immutable flag" term.

* Add an item in the open issues tracking: Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema so that there is a single source of truth?

v05 - v06

* Remove immutable BGP AS number case

* Fix nits

v04 - v05

* Emphasized that the proposal tries to formally document existing allowed behavior

* Reword the abstract and introduction sections;

* Restructure the document;

* Simplified the interface example in Appendix;

* Add immutable BGP AS number and peer-type configuration example.

* Added temporary section in Appendix B about list of existing non-standard solutions

* Clarified inheritance of immutability

* Clarified that this draft is not dependent on the existence of the <system> datastore.

v03 - v04

* Clarify how immutable flag interacts with NACM mechanism.

v02 - v03

* rephrase and avoid using "server MUST reject" statement, and try to clarify that this documents aims to provide visibility into existing immutable behavior;

* Add a new section to discuss the inheritance of immutability;

* Clarify that deletion to an immutable node in <running> which is instantiated in <system> and copied into <running> should always be allowed;

* Clarify that write access restriction due to general YANG rules has no need to be marked as immutable.

* Add an new section named "Acknowledgements";

* editoral changes.

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

Appendix D. Open Issues tracking

* Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema so that there is a single source of truth?

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