

Network Working Group  
Internet-Draft  
Updates: 6020, 7950, 8407, 8525 (if approved)  
Intended status: Standards Track  
Expires: 2 September 2024

R. Wilton, Ed.  
Cisco Systems, Inc.  
R. Rahman, Ed.  
Equinix  
B. Lengyel, Ed.  
Ericsson  
J. Clarke  
Cisco Systems, Inc.  
J. Sterne  
Nokia  
1 March 2024

Updated YANG Module Revision Handling  
draft-ietf-netmod-yang-module-versioning-11

Abstract

This document refines the RFC 7950 module update rules. It 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. 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 2 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1.	Introduction . . . . .	3
1.1.	Updates to YANG RFCs . . . . .	4
2.	Terminology and Conventions . . . . .	4
3.	Refinements to YANG revision handling . . . . .	5
3.1.	Updating a YANG module with a new revision . . . . .	6
3.1.1.	Backwards-compatible rules . . . . .	7
3.1.2.	Non-backwards-compatible changes . . . . .	8
3.2.	non-backwards-compatible extension statement . . . . .	8
3.3.	Removing revisions from the revision history . . . . .	8
3.4.	Examples for updating the YANG module revision history . . . . .	9
4.	Guidance for revision selection on imports . . . . .	12
4.1.	Recommending a minimum revision for module imports . . . . .	13
4.1.1.	Module import examples . . . . .	14
5.	New ietf-yang-status-conformance YANG module . . . . .	15
5.1.	Reporting how deprecated and obsolete nodes are handled . . . . .	15
6.	Guidelines for using the YANG module update rules . . . . .	16
6.1.	Guidelines for YANG module authors . . . . .	16
6.1.1.	Making non-backwards-compatible changes to a YANG module . . . . .	17
6.2.	Versioning Considerations for Clients . . . . .	18
7.	Module Versioning Extension YANG Modules . . . . .	18
8.	Security considerations . . . . .	24
8.1.	Security considerations for module revisions . . . . .	24
8.2.	Security considerations for the modules defined in this document . . . . .	25
9.	IANA Considerations . . . . .	25
9.1.	YANG Module Registrations . . . . .	25
9.2.	Guidance for versioning in IANA maintained YANG modules . . . . .	26
10.	References . . . . .	27
10.1.	Normative References . . . . .	27
10.2.	Informative References . . . . .	28
Appendix A.	Examples of changes that are NBC . . . . .	30
Appendix B.	Examples of applying the NBC change guidelines . . . . .	31
B.1.	Removing a data node . . . . .	31
B.2.	Changing the type of a leaf node . . . . .	31

B.3. Reducing the range of a leaf node . . . . .	32
B.4. Changing the key of a list . . . . .	32
B.5. Renaming a node . . . . .	33
Contributors . . . . .	33
Acknowledgments . . . . .	34
Authors' Addresses . . . . .	34

## 1. Introduction

The current YANG [RFC7950] module update rules require that updates of YANG modules preserve strict backwards compatibility. This causes 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 when importing YANG modules. Accepting that non-backwards-compatible changes do sometimes occur -- e.g., for bugfixes -- it is important to have mechanisms to report when these changes occur, and to manage their effect on clients and the broader YANG ecosystem.

Several other documents build on this document 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 defines a YANG extension that tags a YANG artifact with a version identifier 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 four 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.

- \* 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 report how "deprecated" and "obsolete" nodes are handled by a server.
- \* 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 [RFC8407] section 4.7. Section 6 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 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.

However, using revision dates alone to identify revisions of a YANG module versioned with a branched revision history is likely to be confusing because the relationship between module revisions is no longer guaranteed to be chronologically ordered. Instead, for modules that may use a branched revision history, it is RECOMMENDED to use a version identifier, such as the one described in [I-D.ietf-netmod-yang-semver], that better describes the semantic relationship between the revisions.

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 it 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 version identifier 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], could 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 backwards-compatible (BC) to the previous revision of the module. This document introduces a method to indicate that a non-backwards-compatible (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.

New module revisions SHOULD NOT contain NBC changes because they often create problems for clients, however they can be helpful in some scenarios, and hence are discouraged, but allowed. For example:

- \* Bugfixes, particularly where the likely client impact is low or the module is changed to reflect current server behavior.
- \* To mark nodes as obsolete (or remove them), after a suitable deprecation period.

- \* To refine new and unstable modules (or new and unstable nodes within existing, stable modules).
- \* Restructuring a module to add new functionality where the cost of adding the functionality in a BC manner is disproportionate to the expected benefits of greater client backwards compatibility.

A YANG extension, defined in Section 3.2, is used to signal the potential for incompatibility to existing module users and readers.

As per [RFC7950] and [RFC6020], all published revisions of a module are given a new unique revision date.

### 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-date" substatements of an "import" statement, including adding new "revision-date" or "recommended-min-date" substatements, changing the argument of any "revision-date" or "recommended-min-date" substatements, or removing any "revision-date" or "recommended-min-date" 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 since it 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:non-backwards-compatible;
}

revision 2020-08-09 {
  rev:non-backwards-compatible;
}

revision 2020-06-07 {
}

revision 2020-02-10 {
  rev:non-backwards-compatible;
}

revision 2019-10-21 {
}

revision 2019-03-04 {
}

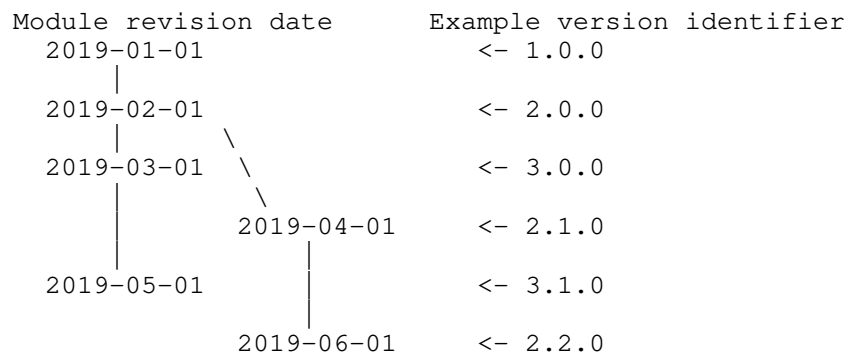
revision 2019-01-02 {
}
```

In the revision history example above (with revision descriptions omitted for clarity), 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. Examples for updating the YANG module revision history

The following diagram, explanation, and module history illustrates how a branched revision history for a YANG module could be represented chronologically. To aid clarity, it makes use of both the "non-backwards-compatible" extension statement, and the "version" extension statement defined in [I-D.ietf-netmod-yang-semver]:

Example YANG module with branched revision history using version identifiers defined in [I-D.ietf-netmod-yang-semver].



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 2019-05-01:

```
module example-module {  
    namespace "urn:example:module";  
    prefix "prefix-name";  
  
    import ietf-yang-revisions { prefix "rev"; }  
    import ietf-yang-semver { prefix "ys"; }  
  
    description  
        "to be completed";  
  
    revision 2019-05-01 {  
        ys:version 3.1.0;  
        description "Add new functionality.";  
    }  
  
    revision 2019-03-01 {  
        ys:version 3.0.0;  
        rev:non-backwards-compatible;  
        description  
            "Add new functionality. Remove some deprecated nodes.";  
    }  
  
    revision 2019-02-01 {  
        ys:version 2.0.0;  
        rev:non-backwards-compatible;  
        description "Apply bugfix to pattern statement";  
    }  
  
    revision 2019-01-01 {  
        ys:version 1.0.0;  
        description "Initial revision";  
    }  
  
    //YANG module definition starts here  
}
```

Example module, revision 2019-06-01:

```
module example-module {  
    namespace "urn:example:module";  
    prefix "prefix-name";  
  
    import ietf-yang-revisions { prefix "rev"; }  
    import ietf-yang-semver { prefix "ys"; }  
  
    description  
        "to be completed";  
  
    revision 2019-06-01 {  
        ys:version 2.2.0;  
        description "Backwards-compatible bugfix to enhancement.";  
    }  
  
    revision 2019-04-01 {  
        ys:version 2.1.0;  
        description "Apply enhancement to older release train.";  
    }  
  
    revision 2019-02-01 {  
        ys:version 2.0.0;  
        rev:non-backwards-compatible;  
        description "Apply bugfix to pattern statement";  
    }  
  
    revision 2019-01-01 {  
        ys:version 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 6.1 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 a 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 module introduces, for modules with a linear revision history that are versioned using revision dates, 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-date"` extension statement, a substatement to the YANG `"import"` statement, to allow for a `"minimum recommended date"` to be documented:

The argument to the `"recommended-min-date"` extension statement is a revision date.

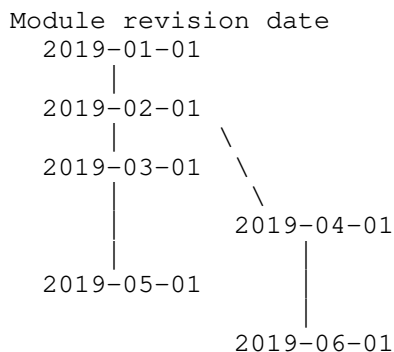
A particular revision of an imported module adheres to an import's `"recommended-min-date"` extension statement if the imported module's revision date is equal to or later than the revision date argument of the `"recommended-min-date"` extension statement in the importing module.

Zero or one `"recommended-min-date"` extension statement is allowed for each parent `"import"` statement.

Adding, modifying or removing a "recommended-min-date" extension statement is a BC change.

#### 4.1.1. Module import examples

Consider the example module "example-module" from Section 3.4 that is hypothetically available in the following revisions: 2019-01-01, 2019-02-01, 2019-03-01, 2019-04-01, 2019-05-01 and 2019-06-01. The relationship between the revisions is as before:



##### 4.1.1.1. Example 1

This example recommends module revisions for import whose revision date is or comes after 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 the following revisions: 2019-02-01, 2019-03-01, 2019-04-01, 2019-05-01 and 2019-06-01.

```

import example-module {
  rev:recommended-min-date 2019-02-01;
}
  
```

##### 4.1.1.2. Example 2

This example recommends module revisions for import whose revision date is or comes after 2019-04-01. It includes the following revisions: 2019-04-01, 2019-05-01 and 2019-06-01, even though revision 2019-05-01 may not contain what is desired from 2019-04-01. This shows that "recommended-min-date" is not well suited for a branched revision history, and is most helpful when a module is restricted to a linear chronological development history.

```
import example-module {  
  rev:recommended-min-date 2019-04-01;  
}
```

## 5. New ietf-yang-status-conformance YANG module

This document defines the YANG module, `ietf-yang-status-conformance`, that augments YANG library [RFC8525] with two leafs to indicate how a server implements deprecated and obsolete schema nodes.

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

```
module: ietf-yang-status-conformance  
  augment /yanglib:yang-library/yanglib:schema:  
    +--ro deprecated-nodes-implemented?  boolean  
    +--ro obsolete-nodes-absent?         boolean
```

### 5.1. Reporting how deprecated and obsolete nodes are handled

The `ietf-yang-status-conformance` 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 by the server 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", which allows clients to determine the exact schema used by the server.

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. 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.

### 6.1. Guidelines for YANG module authors

All IETF YANG modules MUST conform to this specification. In particular, sections: Section 3, Section 4, and the guidelines documented in this section.

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 can cause a loss of visibility of when non-backwards-compatible changes were made, and hence it is RECOMMENDED to retain them. 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.

In cases where a revision dependency is helpful for a module import, the "rev:recommended-min-date" extension SHOULD be used 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

#### 6.1.1. 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 accidentally 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.

## 6.2. 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.

## 7. Module Versioning Extension YANG Modules

YANG module with extension statements for annotating NBC changes and importing by revision.

```
<CODE BEGINS> file "ietf-yang-revisions@2024-02-19.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) 2024 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).
```

```
// 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 2024-02-19 {
  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";
}

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 recommended-min-date {

argument revision-date;

description

"Recommends the revision of the module that may be imported to one whose revision date matches or is after the specified revision-date.

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

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

Zero or one 'recommended-min-date' extension statement is allowed for each parent 'import' statement.

A particular revision of an imported module adheres to an import's 'recommended-min-date' extension statement if the imported module's revision date is equal to or later than the revision date argument of the 'recommended-min-date' extension statement in the importing module.

Adding, removing or updating a 'recommended-min-date' 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";

}

}

<CODE ENDS>

YANG module for status conformance

```
<CODE BEGINS> file "ietf-yang-status-conformance@2024-02-14.yang"
module ietf-yang-status-conformance {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-status-conformance";
  prefix ys-conf;

  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 provide an
    indication of how deprecated and obsolete nodes are handled by
    the server.

    Copyright (c) 2024 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.
```

```
revision 2024-02-14 {
  description
    "Initial revision";
  reference
    "XXXX: Updated YANG Module Revision Handling";
}
```

```
augment "/yanglib:yang-library/yanglib:schema" {
  description
    "Augmentations to the ietf-yang-library module to indicate how
    deprecated and obsoleted nodes are handled 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 5.1, 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 5.1, Reporting how deprecated and obsolete nodes
```

```
        are handled";
    }
}
}
<CODE ENDS>
```

## 8. Security considerations

### 8.1. 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.



## 8.2. 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 two boolean leaves that 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].

## 9. IANA Considerations

### 9.1. 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-status-conformance  
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-status-conformance module:

Name: ietf-yang-status-conformance

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-status-conformance

Prefix: ys-conf

Reference: [RFCXXXX]

## 9.2. 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.

## 10. References

### 10.1. Normative References

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

## 10.2. Informative References

## [AddrFamilyReg]

"Address Family Numbers IANA Registry",  
<[https://www.iana.org/assignments/address-family-numbers/  
address-family-numbers.xhtml](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-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-12, 2 October 2023, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-12>>.

## [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-09, 14 January 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-versioning-reqs-09>>.

- [IfTypesReg] "Interface Types (ifType) IANA Registry",  
<<https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>>.
- [IfTypeYang] "iana-if-type YANG Module",  
<<https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams",  
BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,  
<<https://www.rfc-editor.org/info/rfc8340>>.
- [RoutingTypesDecRevision] "2020-12-31 revision of iana-routing-types.yang",  
<<https://www.iana.org/assignments/yang-parameters/iana-routing-types@2020-12-31.yang>>.
- [RoutingTypesYang] "iana-routing-types YANG Module",  
<<https://www.iana.org/assignments/iana-routing-types/iana-routing-types.xhtml>>.
- [SAFIReg] "Subsequent Address Family Identifiers (SAFI) Parameters  
IANA Registry", <<https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml>>.

#### 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 6.1.1.

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.schoenwaelder@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](mailto:rwilton@cisco.com)

Reshad Rahman (editor)  
Equinix  
Email: [reshad@yahoo.com](mailto:reshad@yahoo.com)

Balazs Lengyel (editor)  
Ericsson  
Email: [balazs.lengyel@ericsson.com](mailto:balazs.lengyel@ericsson.com)

Joe Clarke  
Cisco Systems, Inc.  
Email: [jclarke@cisco.com](mailto:jclarke@cisco.com)

Jason Sterne  
Nokia  
Email: [jason.sterne@nokia.com](mailto:jason.sterne@nokia.com)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 5 September 2022

R. Wilton, Ed.  
R. Rahman  
J. Clarke  
Cisco Systems, Inc.  
J. Sterne  
Nokia  
B. Wu, Ed.  
Huawei  
4 March 2022

YANG Packages  
draft-ietf-netmod-yang-packages-03

Abstract

This document defines YANG packages; a versioned organizational structure used to manage schema and conformance of YANG modules as a cohesive set instead of individually.

It describes how packages: are represented on a server, can be defined in offline YANG instance data files, and can be used to define the content schema associated with YANG instance data files.

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

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

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

## Table of Contents

1. Terminology and Conventions . . . . .	3
2. Introduction . . . . .	4
3. Background on YANG packages . . . . .	4
4. Objectives . . . . .	5
5. YANG Package Definition . . . . .	6
5.1. Package definition rules . . . . .	7
5.2. Package versioning . . . . .	8
5.2.1. Updating a package with a new version . . . . .	8
5.2.1.1. Non-Backwards-compatible changes . . . . .	8
5.2.1.2. Backwards-compatible changes . . . . .	8
5.2.1.3. Editorial changes . . . . .	9
5.2.2. YANG Semantic Versioning for packages . . . . .	9
5.3. Package conformance . . . . .	10
5.3.1. Use of YANG semantic versioning . . . . .	10
5.3.2. The relationship between packages and datastores . . . . .	11
5.4. Schema referential completeness . . . . .	12
5.5. Package name scoping and uniqueness . . . . .	13
5.5.1. Globally scoped packages . . . . .	13
5.5.2. Server scoped packages . . . . .	13
5.6. Submodules packages considerations . . . . .	13
5.7. Package tags . . . . .	14
5.8. YANG Package Usage Guidance . . . . .	14
5.8.1. Use of deviations in YANG packages . . . . .	14
5.8.2. Use of features in YANG modules and YANG packages . . . . .	15
5.9. YANG package core definition . . . . .	15
6. Package Instance Data Files . . . . .	17
7. Package Definitions on a Server . . . . .	18
7.1. Package List . . . . .	18
7.2. Tree diagram . . . . .	18
8. YANG Library Package Bindings . . . . .	19
9. YANG packages as schema for YANG instance data document . . . . .	19
10. YANG Modules . . . . .	20
11. Security Considerations . . . . .	37
12. IANA Considerations . . . . .	38
13. Open Questions/Issues . . . . .	39
14. Acknowledgements . . . . .	40
15. References . . . . .	40

15.1. Normative References . . . . .	40
15.2. Informative References . . . . .	42
Appendix A. Examples . . . . .	43
A.1. Example IETF Network Device YANG package . . . . .	43
A.2. Example IETF Basic Routing YANG package . . . . .	45
A.3. Package import conflict resolution example . . . . .	48
Appendix B. Possible alternative solutions . . . . .	51
B.1. Using module tags . . . . .	52
B.2. Using YANG library . . . . .	52
Authors' Addresses . . . . .	53

## 1. Terminology and Conventions

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

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

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

- \* datastore schema

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

- \* data node

- \* schema node

In addition, this document defines the following terminology:

- \* YANG package: a versioned organizational structure used to manage a set of YANG modules that collectively define a package schema. YANG packages are defined in Section 5.

- \* package schema: The combined set of schema nodes defined by a YANG package. Package schema can be used to define datastore schema.

- \* backwards-compatible (BC) change: When used in the context of a YANG module, it follows the definition in Section 3.1.1 of [I-D.ietf-netmod-yang-module-versioning]. When used in the context of a YANG package, it follows the definition in Section 5.2.1.2.

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

## 2. Introduction

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

A YANG package is a versioned organizational structure used to manage a set of YANG modules that collectively define a package schema. For example, a YANG package could contain the set of YANG modules required to implement an L2VPN service on a network device.

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

## 3. Background on YANG packages

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

'The YANG Package Statement' [I-D.bierman-netmod-yang-package] proposed a YANG package mechanism based on new YANG language statements, where a YANG package is defined in a file similar to how YANG modules are defined, and would require enhancements to YANG compilers to understand the new statements used to define packages.

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

The NETMOD WG in general, and the YANG versioning design team in particular, are exploring solutions [I-D.ietf-netmod-yang-solutions] to the YANG versioning requirements, [I-D.ietf-netmod-yang-versioning-reqs]. Solutions to the versioning requirements can be split into several distinct areas. [I-D.ietf-netmod-yang-module-versioning] is focused on YANG

versioning scoped to individual modules. The overall solution must also consider YANG versioning and conformance scoped to sets of modules. YANG packages provide part of the solution for versioning sets of modules.

#### 4. Objectives

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

- \* To provide an alternative, simplified, YANG conformance mechanism. Rather than conformance being performed against a set of individual YANG module revisions, features, and deviations, conformance can be more simply stated in terms of YANG packages, with a set of modifications (e.g. additional modules, deviations, or features).
- \* To allow datastore schema to be specified in a concise way rather than having each server explicitly list all modules, revisions, and features. YANG package definitions can be defined in documents that are available offline, and accessible via a URL, rather than requiring explicit lists of modules to be shared between client and server. Hence, a YANG package must contain sufficient information to allow a client or server to precisely construct the schema associated with the package.
- \* To define a mainly linear versioned history of sets of modules versions that are known to work together. I.e. to help mitigate the problem where a client must manage devices from multiple vendors, and vendor A implements version 1.0.0 of module foo and version 2.0.0 of module bar, and vendor B implements version 2.0.0 of module foo and version 1.0.0 of module bar. For a client, trying to interoperate with multiple vendors, and many YANG modules, finding a consistent lowest common denominator set of YANG module versions may be difficult, if not impossible.

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

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



## 5. YANG Package Definition

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

A YANG package is a versioned organizational structure used to manage a set of YANG modules that collectively define a package schema.

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

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

Each YANG package version, defines:

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

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

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

YANG package definitions are available from the server via augmentations to YANG Library [RFC8525]. Rather than client applications downloading the entire contents of YANG library to

confirm that the server's datastore schema are compatible with the client, they can simply check the names and versions of the packages advertised in YANG library to know what schema to expect in the server datastores.

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

### 5.1. Package definition rules

Packages are defined using the following rules:

1. A YANG package MAY represent a referentially complete set of modules or MAY represent a set of modules with some module import dependencies missing, as described in Section 5.4.
2. Packages definitions are hierarchical. A package can include other packages. Only a single version of a package can be included, and conflicting package includes (e.g. from descendant package includes) MUST be explicitly resolved by indicating which version takes precedence, and which versions are being replaced.
3. YANG packages definitions MAY include modules containing deviation statements, but those deviation statements MUST only be used in an [RFC7950] compatible way to indicate where a server, or class of servers, deviates from a published standard. Deviations MUST NOT be included in a package definition that is part of a published standard. See section Section 5.8.1 for further guidance on the use of deviations in YANG packages.
4. For each module implemented by a package, only a single revision of that module MUST be implemented. Multiple revisions of a module MAY be listed as import-only dependencies.
5. The revision of a module listed in the package 'module' list supersedes any 'implemented' revision of the module listed in an included package module list. The 'replaces-revision' leaf-list is used to indicate which 'implemented' or 'import-only' module revisions are replaced by this module revision. This allows a package to explicitly resolve conflicts between implemented module revisions in included packages.

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

## 5.2. Package versioning

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

### 5.2.1. Updating a package with a new version

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

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

#### 5.2.1.1. Non-Backwards-compatible changes

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

- \* Changing an 'included-package' list entry to select a package version that is non-backwards-compatible to the prior package version, or removing a previously included package.
- \* Changing a 'module' or 'import-only-module' list entry to select a module revision that is non-backwards-compatible to the prior module revision, or removing a previously implemented module.
- \* Removing a feature from the 'mandatory-feature' leaf-list.
- \* Adding, changing, or removing a module containing one or more deviations, that when applied to the target module would create a change that is considered a non-backwards-compatible change to the affected data node in the schema associated with the prior package version.

#### 5.2.1.2. Backwards-compatible changes

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

- \* Changing an 'included-package' list entry to select a package version that is backwards-compatible to the prior package version, or including a new package that does not conflict with any existing included package or module.
- \* Changing a 'module' or 'import-only-module' list entry to select a module revision that is backwards-compatible to the prior module revision, or including a new module to the package definition.
- \* Adding a feature to the 'mandatory-feature' leaf-list.
- \* Adding, changing, or removing a module containing one or more deviations, that when applied to the target module would create a change that is considered a backwards-compatible change to the affected data node in the schema associated with the prior package version.

#### 5.2.1.3. Editorial changes

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

- \* Changing a 'included-package' list entry to select a package version that is classified as an editorial change relative to the prior package version.
- \* Changing a 'module' or 'import-only-module' list entry to select a module revision that is classified as an editorial change relative to the prior module revision.
- \* Any change to any metadata associated with a package definition.

#### 5.2.2. YANG Semantic Versioning for packages

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

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

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

The rules for incrementing the YANG package version number are equivalent to the semantic versioning rules used to version individual YANG modules, defined in section 3.2 of [I-D.ietf-netmod-yang-semver], but use the rules defined previously

in Section 5.2.1 to determine whether a change is classified as non-backwards-compatible, backwards-compatible, or editorial. Where available, the semantic version number of the referenced elements in the package (included packages or modules) can be used to help determine the scope of changes being made.

### 5.3. Package conformance

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

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

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

Package definitions **MAY** modify the schema for directly or hierarchically included packages through the use of different module revisions or module deviations.

#### 5.3.1. Use of YANG semantic versioning

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

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

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

This also has a relevance on servers that are capable of supporting version selection because they need not support every version of a YANG package to ensure good client compatibility. Choosing suitable

minor versions within each major version number should generally be sufficient, particular if they can avoid non-backwards-compatible patch level changes.

### 5.3.2. The relationship between packages and datastores

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

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

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

- \* Any datastores (e.g., conventional configuration datastores) that have exactly the same datastore schema MUST use the same package definitions. This is to avoid, for example, the creation of a 'running-cfg' package and a separate 'intended-cfg' package that have identical schema.

- \* Common package definitions SHOULD be used for those parts of the datastore schema that are common between datastores, when those datastores do not share exactly the same datastore schema. E.g., if a substantial part of the schema is common between the conventional, dynamic, and operational datastores then a single common package can be used to describe the common parts, along with other packages to describe the unique parts of each datastore schema.
- \* YANG modules that do not contain any configuration data nodes SHOULD be included in the package for configuration datastores if that helps unify the package definitions.
- \* The packages for the operational datastore schema MUST include all packages for all configuration datastores, along with any required modules defining deviations to mark unsupported data nodes. The deviations MAY be defined directly in the packages defining the operational datastore schema, or in separate non referentially complete packages.
- \* The schema for a datastore MAY be represented using a single package or as the union of a set of compatible packages, i.e., equivalently to a set of non-conflicting packages being included together in an overarching package definition.

#### 5.4. Schema referential completeness

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

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

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

Referentially incomplete packages can be used, along with locally scoped packages, to represent an update to a device's datastore schema as part of an optional software hot fix. E.g., the base software is made available as a complete globally scoped package. The hot fix is made available as an incomplete globally scoped package. A device's datastore schema can define a local package that implements the base software package updated with the hot fix package.

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

#### 5.5. Package name scoping and uniqueness

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

##### 5.5.1. Globally scoped packages

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

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

##### 5.5.2. Server scoped packages

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

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

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

#### 5.6. Submodules packages considerations

As defined in [RFC7950] and [I-D.ietf-netmod-yang-semver], YANG conformance and versioning is specified in terms of particular revisions of YANG modules rather than for individual submodules.



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

### 5.7. Package tags

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

### 5.8. YANG Package Usage Guidance

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

#### 5.8.1. Use of deviations in YANG packages

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

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

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

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

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

#### 5.8.2. Use of features in YANG modules and YANG packages

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

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

#### 5.9. YANG package core definition

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

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

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

grouping yang-pkg-identification-leafs
  +-- name          pkg-name
  +-- version       pkg-version

grouping yang-pkg-instance
  +-- name          pkg-name
  +-- version       pkg-version
  +-- timestamp?   yang:date-and-time
  +-- organization? string
  +-- contact?     string
  +-- description? string
  +-- reference?   string
  +-- complete?    boolean
  +-- local?       boolean
  +-- tag*         tags:tag
  +-- mandatory-feature* scoped-feature
  +-- included-package* [name version]
  |   +-- name          pkg-name
  |   +-- version       pkg-version
  |   +-- replaces-version* pkg-version
  |   +-- location*    inet:uri
  +-- module* [name]
  |   +-- name          yang:yang-identifier
  |   +-- revision?    rev:revision-date-or-label
  |   +-- replaces-revision* rev:revision-date-or-label
  |   +-- namespace?  inet:uri
  |   +-- location*    inet:uri
  |   +-- submodule* [name]
  |   |   +-- name?      yang:yang-identifier
  |   |   +-- revision  yang:revision-identifier
  |   |   +-- location* inet:uri
  +-- import-only-module* [name revision]
  |   +-- name?        yang:yang-identifier
  |   +-- revision?    rev:revision-date-or-label
  |   +-- replaces-revision* rev:revision-date-or-label
  |   +-- namespace?  inet:uri
  |   +-- location*    inet:uri
  |   +-- submodule* [name]
  |   |   +-- name?      yang:yang-identifier
  |   |   +-- revision  yang:revision-identifier
  |   |   +-- location* inet:uri
```

## 6. Package Instance Data Files

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

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

1. The file SHOULD be encoded in JSON.
2. The name of the file SHOULD follow the format "<package-name>@<version>.json".
3. The package name MUST be specified in both the instance-data-set 'name' and package 'name' leafs.
4. The 'description' field of the instance-data-set SHOULD be "YANG package definition".
5. The 'timestamp', 'organization', 'contact' fields are defined in both the instance-data-set metadata and the YANG package metadata. Package definitions SHOULD only define these fields as part of the package definition. If any of these fields are populated in the instance-data-set metadata then they MUST contain the same value as the corresponding leaves in the package definition.
6. The 'revision' list in the instance data file SHOULD NOT be used, since versioning is handled by the package definition.
7. The instance data file for each version of a YANG package SHOULD be made available at one of more locations accessible via URLs. If one of the listed locations defines a definitive reference implementation for the package definition then it MUST be listed as the first entry in the list.

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

```
module: ietf-yang-package

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

## 7. Package Definitions on a Server

### 7.1. Package List

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

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

### 7.2. Tree diagram

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

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

## 8. YANG Library Package Bindings

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

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

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

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

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

## 9. YANG packages as schema for YANG instance data document

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

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

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

```
  augment-structure /yid:instance-data-set/yid:content-schema/yid:content-schema-spec:
```

```
    +--: (pkg-schema)
      +-- pkg-schema
        +-- name          pkg-name
        +-- version      pkg-version
        +-- location*    inet:uri
```

## 10. YANG Modules

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

```
<CODE BEGINS>
  file "ietf-yang-package-types#0.3.0-draft-ietf-netmod-yang-packages-03.yang"
  module ietf-yang-package-types {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-yang-package-types";
    prefix pkg-types;

    import ietf-yang-revisions {
      prefix rev;
      reference
        "XXXX: Updated YANG Module Revision Handling";
    }
    import ietf-yang-types {
      prefix yang;
      rev:revision-or-derived "2019-07-21";
      reference
        "RFC 6991bis: Common YANG Data Types.";
    }
    import ietf-inet-types {
      prefix inet;
      rev:revision-or-derived "2013-07-15";
      reference
        "RFC 6991: Common YANG Data Types.";
    }
    import ietf-module-tags {
      prefix tags;
      reference
        "RFC 8819: YANG Module Tags.";
    }

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

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

Author:   Rob Wilton
         <mailto:rwilton@cisco.com>;
description
"This module provides type and grouping definitions for YANG
packages.

Copyright (c) 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; see
the RFC itself for full legal notices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
'MAY', and 'OPTIONAL' in this document are to be interpreted as
described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
they appear in all capitals, as shown here.";

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

revision 2022-03-04 {
  rev:revision-label 0.3.0-draft-ietf-netmod-yang-packages-03;
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}

/*
 * Typedefs
 */

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



```
    }

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

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

    typedef scoped-feature {
      type string {
        pattern '[a-zA-Z_][a-zA-Z0-9\-\_\.]*:[a-zA-Z_][a-zA-Z0-9\-\_\.]*';
      }
      description
        "Represents a feature name scoped to a particular module,
        identified as the '<module-name>:<feature-name>', where both
        <module-name> and <feature-name> are YANG identifier strings,
        as defined by Section 12 or RFC 6020.";
      reference
        "RFC XXXX, YANG Packages.";
    }

    /*
     * Groupings
     */

    grouping yang-pkg-identification-leafs {
      description
        "Parameters for identifying a specific version of a YANG
        package";
      leaf name {
        type pkg-name;
        mandatory true;
        description
          "The YANG package name.";
      }
      leaf version {
        type pkg-version;
        mandatory true;
        description
          "Uniquely identifies a particular version of a YANG package."
      }
    }
  }
}
```

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

grouping yang-pkg-instance {
  description
    "Specifies the data node for a full YANG package instance
    represented either on a server or as a YANG instance data
    document.";
  uses yang-pkg-identification-leafs;
  leaf timestamp {
    type yang:date-and-time;
    description
      "An optional timestamp for when this package was created.
      This does not need to be unique across all versions of a
      package.";
  }
  leaf organization {
    type string;
    description
      "Organization responsible for this package";
  }
  leaf contact {
    type string;
    description
      "Contact information for the person or organization to whom
      queries concerning this package should be sent.";
  }
  leaf description {
    type string;
    description
      "Provides a description of the package";
  }
  leaf reference {
    type string;
    description
      "Allows for a reference for the package";
  }
  leaf complete {
    type boolean;
    default "true";
    description
      "Indicates whether the schema defined by this package is
      referentially complete. I.e. all module imports can be
      resolved to a module explicitly defined in this package or
      one of the included packages.";
  }
}
```

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

    Local packages can be used when the schema for the device
    can be changed at runtime through the addition or removal of
    software packages, or hot fixes.";
}
leaf-list tag {
  type tags:tag;
  description
    "Tags associated with a YANG package.  Module tags defined in
    XXX, ietf-netmod-module-tags can be used here but with the
    modification that the tag applies to the entire package
    rather than a specific module.  See the IANA 'YANG Module
    Tag Prefix' registry for reserved prefixes and the IANA
    'YANG Module IETF Tag' registry for IETF standard tags.";
}
leaf-list mandatory-feature {
  type scoped-feature;
  description
    "Lists features from any modules included in the package that
    MUST be supported by any server implementing the package.

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

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

    Features are identified using <module-name>:<feature-name>";
}
list included-package {
  key "name version";
  description
    "An entry in this list represents a package that is included
    as part of the package definition, or an indirectly included
    package that is changed in a non backwards compatible way.

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

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

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

```
reference
"XXX";
uses yang-pkg-identification-leafs;
leaf-list replaces-version {
  type pkg-version;
  description
    "Gives the version of an included package version that
     is replaced by this included package version.";
}
leaf-list location {
  type inet:uri;
  description
    "Contains a URL that represents where an instance data file
     for this YANG package can be found.

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

    If multiple locations are provided, then the first
    location in the leaf-list MUST be the definitive location
    that uniquely identifies this package";
}
}
list module {
  key "name";
  description
    "An entry in this list represents a module that must be
     implemented by a server implementing this package, as per
     RFC 7950 section 5.6.5, with a particular set of supported
     features and deviations.

     A entry in this list overrides any module revision
     'implemented' by an included package. Any replaced module
     revision SHOULD also be listed in the 'replaces-revision'
     list.";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";
  leaf name {
    type yang:yang-identifier;
    mandatory true;
```

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

      If no revision statement is present in the YANG module,
      this leaf is not instantiated.";
  }
  leaf-list replaces-revision {
    type rev:revision-date-or-label;
    description
      "Gives the revision of an module (implemented or
      import-only) defined in an included package that is
      replaced by this implemented module revision.";
  }
  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.";
  }
  leaf-list location {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema resource
      for this module.

      This leaf will only be present if there is a URL available
      for retrieval of the schema for this entry.";
  }
  list submodule {
    key "name";
    description
      "Each entry represents one submodule within the
      parent module.";
    leaf name {
      type yang:yang-identifier;
      description
        "The YANG submodule name.";
    }
    leaf revision {
      type rev:revision-date-or-label;
      mandatory true;
      description
        "The YANG submodule revision date or revision-label.
```

```
        If the parent module include statement for this submodule
        includes a revision date then it MUST match the revision
        date specified here or it MUST match the revision-date
        associated with the revision-label specified here.";
    }
    leaf-list location {
        type inet:uri;
        description
            "Contains a URL that represents the YANG schema resource
            for this submodule.

            This leaf will only be present if there is a URL
            available for retrieval of the schema for this entry.";
    }
}
list import-only-module {
    key "name revision";
    description
        "An entry in this list indicates that the server imports
        reusable definitions from the specified revision of the
        module, but does not implement any protocol accessible
        objects from this revision.

        Multiple entries for the same module name MAY exist. This
        can occur if multiple modules import the same module, but
        specify different revision-dates in the import statements.";
    leaf name {
        type yang:yang-identifier;
        description
            "The YANG module name.";
    }
    leaf revision {
        type rev:revision-date-or-label;
        description
            "The YANG module revision date or revision-label.

            If no revision statement is present in the YANG module,
            this leaf is not instantiated.";
    }
    leaf-list replaces-revision {
        type rev:revision-date-or-label;
        description
            "Gives the revision of an import-only-module defined in an
            included package that is replaced by this
            import-only-module revision.";
    }
    leaf namespace {
```



```
<CODE BEGINS>
file "ietf-yang-package-instance#0.3.0-draft-ietf-netmod-yang-packages-03.ya
ng"
module ietf-yang-package-instance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-package-instance";
  prefix pkg-inst;

  import ietf-yang-revisions {
    prefix rev;
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }
  import ietf-yang-package-types {
    prefix pkg-types;
    rev:revision-or-derived "0.2.0";
    reference
      "RFC XXX: this RFC.";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions.";
  }

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

    Author: Rob Wilton
            <mailto:rwilton@cisco.com>";
  description
    "This module provides a definition of a YANG package, which is
    used as the content schema for an YANG instance data document specifying
    a YANG package.

    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; see
```



the RFC itself for full legal notices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

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

```
revision 2022-03-04 {
  rev:revision-label 0.3.0-draft-ietf-netmod-yang-packages-03;
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}
```

```
/*
 * Top-level structure
 */
sx:structure "package" {
  description
    "Defines the YANG package structure for use in a YANG instance
    data document.";
  uses pkg-types:yang-pkg-instance;
}
}
```

<CODE ENDS>

<CODE BEGINS>

```
file "ietf-yang-packages#0.3.0-draft-ietf-netmod-yang-packages-03.yang"
module ietf-yang-packages {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-packages";
  prefix pkgs;

  import ietf-yang-revisions {
    prefix rev;
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }
  import ietf-yang-package-types {
    prefix pkg-types;
    rev:revision-or-derived "0.2.0";
```

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

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

  Author: Rob Wilton
          <mailto:rwilton@cisco.com>";
description
  "This module defines YANG packages on a server implementation.

  Copyright (c) 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; see
  the RFC itself for full legal notices.

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
  'MAY', and 'OPTIONAL' in this document are to be interpreted as
  described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
  they appear in all capitals, as shown here.";

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

revision 2022-03-04 {
  rev:revision-label 0.3.0-draft-ietf-netmod-yang-packages-03;
  description
```

```
    "Initial revision";
  reference
    "RFC XXXX: YANG Packages";
}

/*
 * Groupings
 */

grouping yang-pkg-ref {
  description
    "Defines the leaves used to reference a single YANG package";
  leaf name {
    type leafref {
      path "/pkgs:packages/pkgs:package/pkgs:name";
    }
    description
      "The name of the references package.";
  }
  leaf version {
    type leafref {
      path "/pkgs:packages"
        + '/pkgs:package[pkgs:name = current()/../name]'
        + "/pkgs:version";
    }
    description
      "The version of the referenced package.";
  }
}

grouping yang-ds-pkg-ref {
  description
    "Defines the list used to reference a set of YANG packages that
    collectively represent a datastore schema.";
  list package {
    key "name version";
    description
      "Identifies the YANG packages that collectively defines the
      schema for the associated datastore.

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

      All of the referenced packages must be compatible with no
      conflicting module versions or dependencies.";
    uses yang-pkg-ref;
  }
}
```

```
    }  
  
    /*  
    * Top level data nodes.  
    */  
  
    container packages {  
        config false;  
        description  
            "All YANG package definitions";  
        list package {  
            key "name version";  
            description  
                "YANG package instance";  
            uses pkg-types:yang-pkg-instance;  
            leaf-list location {  
                type inet:uri;  
                description  
                    "Contains a URL that represents where an instance data file  
                    for this YANG package can be found.  
  
                    This leaf will only be present if there is a URL available  
                    for retrieval of the schema for this entry.  
  
                    If multiple locations are provided, then the first  
                    location in the leaf-list MUST be the definitive location  
                    that uniquely identifies this package";  
            }  
        }  
    }  
}  
}  
}  
<CODE ENDS>  
  
<CODE BEGINS>  
file "ietf-yl-package#0.3.0-draft-ietf-netmod-yang-packages-03.yang"  
module ietf-yl-packages {  
    yang-version 1.1;  
    namespace "urn:ietf:params:xml:ns:yang:ietf-yl-packages";  
    prefix yl-pkgs;  
  
    import ietf-yang-revisions {  
        prefix rev;  
        reference  
            "XXXX: Updated YANG Module Revision Handling";  
    }  
    import ietf-yang-packages {  
        prefix pkgs;  
        rev:revision-or-derived "0.2.0";  
    }  
}
```

```
reference
  "RFC XXX: YANG Packages.";
}
import ietf-yang-library {
  prefix yanglib;
  rev:revision-or-derived "2019-01-04";
  reference
    "RFC 8525: YANG Library";
}

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

  Author: Rob Wilton
         <mailto:rwilton@cisco.com>";
description
  "This module provides defined augmentations to YANG library to
  allow a server to report YANG package information.

  Copyright (c) 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).
```

```
    rev:revision-label 0.3.0-draft-ietf-netmod-yang-packages-03;
    description
      "Initial revision";
    reference
      "RFC XXXX: YANG Packages";
  }

/*
 * Augmentations
 */

augment "/yanglib:yang-library/yanglib:schema" {
  description
    "Allow datastore schema to be related to a set of YANG
    packages";
  uses pkgs:yang-ds-pkg-ref;
}
}
<CODE ENDS>

<CODE BEGINS>
file "ietf-yang-inst-data-pkg#0.3.0-draft-ietf-netmod-yang-packages-03.yang"
module ietf-yang-inst-data-pkg {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-inst-data-pkg";
  prefix yid-pkg;

  import ietf-yang-revisions {
    prefix rev;
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }
  import ietf-yang-package-types {
    prefix pkg-types;
    rev:revision-or-derived "0.2.0";
    reference
      "RFC XXX: this RFC.";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions.";
  }
  import ietf-yang-instance-data {
    prefix yid;
    reference
      "RFC 9195: A File Format for YANG Instance Data.";
  }
}
```

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

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

  Author: Rob Wilton
         <mailto:rwilton@cisco.com>";
description
  "The module augments ietf-yang-instance-data to allow package
  definitions to be used to define content schema in YANG instance data
  documents.

  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; see
  the RFC itself for full legal notices.

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
  'MAY', and 'OPTIONAL' in this document are to be interpreted as
  described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
  they appear in all capitals, as shown here.";

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

revision 2022-03-04 {
  rev:revision-label 0.3.0-draft-ietf-netmod-yang-packages-03;
  description
    "Initial revision";
  reference
```

```
    "RFC XXXX: YANG Packages";
  }

  /*
  * Augmentations
  */
  sx:augment-structure "/yid:instance-data-set/yid:content-schema/yid:content-
schema-spec" {
  description
  "Add package reference to instance data set schema
  specification";
  case pkg-schema {
  container pkg-schema {
    uses pkg-types:yang-pkg-identification-leafs;
    leaf-list location {
      type inet:uri;
      description
      "Contains a URL that represents where an instance data
      file for this YANG package can be found.

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

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

## 11. Security Considerations

The YANG modules specified in this document defines a schema for data that is accessed by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

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



Similarly to YANG library [I-D.ietf-netconf-rfc7895bis], some of the readable data nodes in these YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes.

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

As per the YANG library security considerations, the module, revision information in YANG packages may help an attacker identify the server capabilities and server implementations with known bugs since the set of YANG modules supported by a server may reveal the kind of device and the manufacturer of the device. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the YANG packages information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

## 12. IANA Considerations

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

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

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

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

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

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

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

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

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

Name: ietf-yang-package-types.yang  
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-package-types.yang  
Prefix: pkg-types  
Reference: RFC XXXX

Name: ietf-yang-package-instance.yang  
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-package-instance.yang  
Prefix: pkg-inst  
Reference: RFC XXXX

Name: ietf-yang-packages.yang  
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-packages.yang  
Prefix: pkgs  
Reference: RFC XXXX

Name: ietf-yl-packages.yang  
Namespace: urn:ietf:params:xml:ns:yang:ietf-yl-packages.yang  
Prefix: yl-pkgs  
Reference: RFC XXXX

Name: ietf-yang-inst-data-pkg.yang  
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-inst-data-pkg.yang  
Prefix: yid-pkg  
Reference: RFC XXXX

### 13. Open Questions/Issues

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

## 14. Acknowledgements

Feedback helping shape this document has kindly been provided by Andy Bierman, James Cumming, Mahesh Jethanandani, Balazs Lengyel, Ladislav Lhotka, and Jan Lindblad.

## 15. References

### 15.1. Normative References

[I-D.ietf-netconf-rfc7895bis]

Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", Work in Progress, Internet-Draft, draft-ietf-netconf-rfc7895bis-07, 17 October 2018, <<https://www.ietf.org/archive/id/draft-ietf-netconf-rfc7895bis-07.txt>>.

[I-D.ietf-netmod-module-tags]

Hopps, C., Berger, L., and D. Bogdanovic, "YANG Module Tags", Work in Progress, Internet-Draft, draft-ietf-netmod-module-tags-10, 29 February 2020, <<https://www.ietf.org/archive/id/draft-ietf-netmod-module-tags-10.txt>>.

[I-D.ietf-netmod-yang-data-ext]

Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-data-ext-05, 9 December 2019, <<https://www.ietf.org/archive/id/draft-ietf-netmod-yang-data-ext-05.txt>>.

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

Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-instance-file-format-21, 8 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-netmod-yang-instance-file-format-21.txt>>.

[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-05, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-netmod-yang-module-versioning-05.txt>>.

- [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-06, 30 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-netmod-yang-semver-06.txt>>.
- [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://www.ietf.org/archive/id/draft-ietf-netmod-yang-solutions-01.txt>>.
- [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://www.ietf.org/archive/id/draft-ietf-netmod-yang-ver-selection-00.txt>>.
- [I-D.ietf-netmod-yang-versioning-reqs]  
Clarke, J., "YANG Module Versioning Requirements", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-versioning-reqs-06, 6 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-netmod-yang-versioning-reqs-06.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.
- [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/info/rfc8791>>.

## 15.2. Informative References

- [I-D.bierman-netmod-yang-package]  
Bierman, A., "The YANG Package Statement", Work in Progress, Internet-Draft, draft-bierman-netmod-yang-package-00, 6 July 2015, <<https://www.ietf.org/archive/id/draft-bierman-netmod-yang-package-00.txt>>.

[I-D.ietf-netmod-artwork-folding]

Watson, K., Auerswald, E., Farrel, A., and Q. Wu,  
"Handling Long Lines in Content of Internet-Drafts and  
RFCs", Work in Progress, Internet-Draft, draft-ietf-  
netmod-artwork-folding-12, 20 January 2020,  
<[https://www.ietf.org/archive/id/draft-ietf-netmod-  
artwork-folding-12.txt](https://www.ietf.org/archive/id/draft-ietf-netmod-artwork-folding-12.txt)>.

[openconfigsemver]

"Semantic Versioning for OpenConfig Models",  
<<http://www.openconfig.net/docs/semver/>>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module  
Classification", RFC 8199, DOI 10.17487/RFC8199, July  
2017, <<https://www.rfc-editor.org/info/rfc8199>>.

## Appendix A. Examples

This section provides various examples of YANG packages, and as such this text is non-normative. The purpose of the examples is to only illustrate the file format of YANG packages, and how package dependencies work. It does not imply that such packages will be defined by IETF, or which modules would be included in those packages even if they were defined. For brevity, the examples exclude namespace declarations, and use a shortened URL of "tiny.cc/ietf-yang" as a replacement for "<https://raw.githubusercontent.com/YangModels/yang/master/standard/ietf/RFC>".

### A.1. Example IETF Network Device YANG package

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

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

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

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

{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-ietf-network-device-pkg",
    "content-schema": {
      "pkg-schema": {
        "name": "ietf-yang-package-defn-pkg",
        "version": "0.1.0"
      }
    },
    "description": "YANG package definition",
    "content-data": {
      "ietf-yang-package-instance:yang-package": {
        "name": "example-ietf-network-device-pkg",
        "version": "1.1.2",
        "timestamp": "2018-12-13T17:00:00Z",
        "organization": "IETF NETMOD Working Group",
        "contact" : "WG Web: <http://tools.ietf.org/wg/netmod/>, \
                    WG List: <mailto:netmod@ietf.org>",
        "description": "Example IETF network device YANG package.\
          \
          This package defines a small sample set of \
          YANG modules that could represent the basic set of \
          modules that a standard network device might be expected \
          to support.",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "location": [ "file://example.org/yang/packages/\
                    ietf-network-device@v1.1.2.json" ],
        "module": [
          {
            "name": "iana-crypt-hash",
            "revision": "2014-08-06",
            "location": [ "https://tiny.cc/ietf-yang/\
                    iana-crypt-hash%402014-08-06.yang" ],
          },
          {
            "name": "ietf-system",
            "revision": "2014-08-06",
            "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-system%402014-08-06.yang" ],
          },
          {
            "name": "ietf-interfaces",
            "revision": "2018-02-20",
            "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-interfaces%402018-02-20.yang" ],
          }
        ]
      }
    }
  }
}
```

```

    },
    {
      "name": "ietf-netconf-acm",
      "revision": "2018-02-14",
      "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-netconf-acm%402018-02-14.yang" ],
    },
    {
      "name": "ietf-key-chain",
      "revision": "2017-06-15",
      "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-key-chain@2017-06-15.yang" ],
    },
    {
      "name": "ietf-ip",
      "revision": "2018-02-22",
      "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-ip%402018-02-22.yang" ],
    }
  ],
  "import-only-module": [
    {
      "name": "ietf-yang-types",
      "revision": "2013-07-15",
      "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-yang-types%402013-07-15.yang" ],
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "location": [ "https://tiny.cc/ietf-yang/\
                    ietf-inet-types%402013-07-15.yang" ],
    }
  ]
}
}
}
}
}
<CODE ENDS>

```

#### A.2. Example IETF Basic Routing YANG package

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



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

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

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

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-ietf-routing-pkg",
    "content-schema": {
      "pkg-schema": {
        "name": "ietf-yang-package-defn-pkg",
        "version": "0.1.0"
      }
    },
    "description": "YANG package definition",
    "content-data": {
      "ietf-yang-package-instance:yang-package": {
        "name": "example-ietf-routing",
        "version": "1.3.1",
        "timestamp": "2018-12-13T17:00:00Z",
        "description": "This package defines a small sample set of \
          IETF routing YANG modules that could represent the set of \
          IETF routing functionality that a basic IP network device \
          might be expected to support.",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "imported-packages": [
          {
            "name": "ietf-network-device",
            "version": "1.1.2",
            "location": [ "http://example.org/yang/packages/\
              ietf-network-device@v1.1.2.json" ],
          }
        ],
        "module": [
          {
            "name": "ietf-routing",
            "revision": "2018-03-13",
```

```
    "location": [ "https://tiny.cc/ietf-yang/\n                  ietf-routing@2018-03-13.yang" ],\n  },\n  {\n    "name": "ietf-ipv4-unicast-routing",\n    "revision": "2018-03-13",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  ietf-ipv4-unicast-routing@2018-03-13.yang" ],\n  },\n  {\n    "name": "ietf-ipv6-unicast-routing",\n    "revision": "2018-03-13",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  ietf-ipv6-unicast-routing@2018-03-13.yang" ],\n  },\n  {\n    "name": "ietf-isis",\n    "revision": "2018-12-11",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  " ],\n  },\n  {\n    "name": "ietf-interfaces-common",\n    "revision": "2018-07-02",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  " ],\n  },\n  {\n    "name": "ietf-if-l3-vlan",\n    "revision": "2017-10-30",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  " ],\n  },\n  {\n    "name": "ietf-routing-policy",\n    "revision": "2018-10-19",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  " ],\n  },\n  {\n    "name": "ietf-bgp",\n    "revision": "2018-05-09",\n    "location": [ "https://tiny.cc/ietf-yang/\n                  " ],\n  },\n  {\n    "name": "ietf-access-control-list",\n    "revision": "2018-11-06",
```

```

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

```

### A.3. Package import conflict resolution example

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

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

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-import-1-pkg",
    "content-schema": {
      "pkg-schema": {
        "name": "ietf-yang-package-defn-pkg",
        "version": "0.1.0"
      }
    },
    "description": "First imported example package",
    "content-data": {
      "ietf-yang-package-instance:yang-package": {
        "name": "example-import-1",
        "version": "1.0.0",
        "reference": "XXX, draft-rwilton-netmod-yang-packages",
        "revision-date": "2018-01-01",
        "module": [
          {
            "name": "example-module-A",
            "revision": "1.0.0"
          },
          {
            "name": "example-module-B",
            "revision": "1.0.0"
          }
        ],
        "import-only-module": [
          {
            "name": "example-types-module-C",
            "revision": "2018-01-01"
          },
          {
            "name": "example-types-module-D",
            "revision": "2018-01-01"
          }
        ]
      }
    }
  }
}
```

```
    }
  }
  {
    "ietf-yang-instance-data:instance-data-set": {
      "name": "example-import-2-pkg",
      "content-schema": {
        "pkg-schema": {
          "name": "ietf-yang-package-defn-pkg",
          "version": "0.1.0"
        }
      },
      "description": "Second imported example package",
      "content-data": {
        "ietf-yang-package:yang-package": {
          "name": "example-import-2",
          "version": "2.0.0",
          "reference": "XXX, draft-rwilton-netmod-yang-packages",
          "revision-date": "2018-11-26",
          "module": [
            {
              "name": "example-module-A",
              "revision": "1.2.3"
            },
            {
              "name": "example-module-E",
              "revision": "1.1.0"
            }
          ],
          "import-only-module": [
            {
              "name": "example-types-module-C",
              "revision": "2018-11-26"
            },
            {
              "name": "example-types-module-D",
              "revision": "2018-11-26"
            }
          ]
        }
      }
    }
  }
  {
    "ietf-yang-instance-data:instance-data-set": {
      "name": "example-3-pkg",
      "content-schema": {
```

```
    "pkg-schema": {
      "name": "ietf-yang-package-defn-pkg",
      "version": "0.1.0"
    }
  },
  "description": "Importing example package",
  "content-data": {
    "ietf-yang-package:yang-package": {
      "name": "example-3",
      "version": "1.0.0",
      "reference": "XXX, draft-rwilton-netmod-yang-packages",
      "revision-date": "2018-11-26",
      "included-package": [
        {
          "name": "example-import-1",
          "version": "1.0.0"
        },
        {
          "name": "example-import-2",
          "version": "2.0.0"
        }
      ],
      "module": [
        {
          "name": "example-module-A",
          "revision": "1.2.3"
        }
      ],
      "import-only-module": [
        {
          "name": "example-types-module-C",
          "revision": "2018-11-26",
          "replaces-revision": [ "2018-01-01 " ]
        }
      ]
    }
  }
}
```

## Appendix B. Possible alternative solutions

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

### B.1. Using module tags

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

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

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

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

### B.2. Using YANG library

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

1. Packages allow schema to be built in a hierarchical fashion. [I-D.ietf-netconf-rfc7895bis] only allows one layer of hierarchy (using module sets), and there must be no conflicts between module revisions in different module-sets.
2. Packages can be made available off the box, with a well defined unique name, avoiding the need for clients to download, and construct/check the entire schema for each datastore. YANG library's use of a 'content-id' is unique only to the device that generated them.
3. Packages may be versioned using a semantic versioning scheme, YANG library does not provide a schema level semantic version number.
4. For a YANG library instance data file to contain the necessary information, it probably needs both YANG library and various augmentations (e.g. to include each module's semantic version

number), unless a new version of YANG library is defined containing this information. The module definition for a YANG package is specified to contain all of the necessary information to solve the problem without augmentations

5. YANG library is designed to publish information about the modules, datastores, and datastore schema used by a server. The information required to construct an off box schema is not precisely the same, and hence the definitions might deviate from each other over time.

#### Authors' Addresses

Robert Wilton (editor)  
Cisco Systems, Inc.  
Email: [rwilton@cisco.com](mailto:rwilton@cisco.com)

Reshad Rahman  
Cisco Systems, Inc.  
Email: [rrahman@cisco.com](mailto:rrahman@cisco.com)

Joe Clarke  
Cisco Systems, Inc.  
Email: [jclarke@cisco.com](mailto:jclarke@cisco.com)

Jason Sterne  
Nokia  
Email: [jason.sterne@nokia.com](mailto:jason.sterne@nokia.com)

Bo Wu (editor)  
Huawei  
Email: [lane.wubo@huawei.com](mailto:lane.wubo@huawei.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.

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 September 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Key Issues . . . . .	3
1.1. On-wire vs Schema analysis . . . . .	3
1.2. error-tags, error messages, and other error statements .	4
1.3. Comparison on module or full schema (YANG artifact, arbitrary blob. Questions . . . . .	4
2. Open Issues . . . . .	4
2.1. Override/per-node tags . . . . .	5
2.2. Separate rules for config vs state . . . . .	5
2.3. Tool/report verbosity . . . . .	5
2.4. sub-modules . . . . .	5
2.5. Write algorithm in pseudo code or just describe the rules/ goals in text? . . . . .	5
2.6. Categories in the report: bc, nbc, potentially-nbc, editorial. Allow filtering in the draft without defining it? . . . . .	5
2.7. Only for YANG 1.1? . . . . .	5
2.8. renamed-from . . . . .	5
3. Tool options . . . . .	5
4. Introduction . . . . .	6
5. Terminology and Conventions . . . . .	7
6. Generic YANG schema tree comparison algorithm . . . . .	8
6.1. YANG module revision scope extension annotations . . . . .	9
6.2. Node compatibility extension statements . . . . .	9
7. YANG module comparison algorithm . . . . .	13
8. YANG schema comparison algorithms . . . . .	13
8.1. Standard YANG schema comparison algorithm . . . . .	13
8.2. Filtered YANG schema comparison algorithm . . . . .	14
9. Comparison tooling . . . . .	15
10. Module Versioning Extension YANG Modules . . . . .	15
11. Contributors . . . . .	21
12. Security Considerations . . . . .	22
13. IANA Considerations . . . . .	22
13.1. YANG Module Registrations . . . . .	22
14. References . . . . .	22
14.1. Normative References . . . . .	22
14.2. Informative References . . . . .	23
Authors' Addresses . . . . .	24

## 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 consensus 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 compatibility 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 subtree, 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 compatibility statements. A simple copy-n-paste should not be used. The node compatibility 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 compatibility 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 compatibility statement is the authoritative classification of the backwards compatibility of the change to the schema node in the specified revision. This allows a human author to explicitly communicate the compatibility 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 compatibility 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 compatibility statements that reference that revision MUST also be removed. Conversely, node compatibility statements MUST NOT be removed unless the associated revision statement in the revision history is removed.

If a node compatibility 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 compatibility 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 compatibility 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 compatibility 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 particularly 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 have a single node compatibility 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 compatibility 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 is 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 have a single node compatibility 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 compatibility 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 is 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 have a single node compatibility 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 compatibility 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 is 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 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 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, 8525, 7950 (if approved)  
Intended status: Standards Track  
Expires: 19 September 2024

J. Clarke, Ed.  
R. Wilton, Ed.  
Cisco Systems, Inc.  
R. Rahman  
Equinix  
B. Lengyel  
Ericsson  
J. Sterne  
Nokia  
B. Claise  
Huawei  
18 March 2024

YANG Semantic Versioning  
draft-ietf-netmod-yang-semver-15

Abstract

This document specifies a YANG extension along with guidelines for applying an extended set of semantic versioning rules to revisions of YANG artifacts (e.g., modules and packages). Additionally, this document defines a YANG extension for controlling module imports based on these modified semantic versioning rules. This document updates RFCs 7950, 8407, and 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 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
2. Examples of How Versioning Is Applied To YANG Module Revisions . . . . .	3
3. Terminology and Conventions . . . . .	4
4. YANG Semantic Versioning . . . . .	4
4.1. Relationship Between SemVer and YANG Semver . . . . .	5
4.2. YANG Semantic Version Extension . . . . .	5
4.3. YANG Semver Pattern . . . . .	5
4.4. Semantic Versioning Scheme for YANG Artifacts . . . . .	6
4.4.1. Branching Limitations with YANG Semver . . . . .	8
4.4.2. YANG Semver with submodules . . . . .	9
4.4.3. Examples for YANG semantic versions . . . . .	9
4.5. YANG Semantic Version Update Rules . . . . .	11
4.6. Examples of the YANG Semver Label . . . . .	13
4.6.1. Example Module Using YANG Semver . . . . .	13
4.6.2. Example of Package Using YANG Semver . . . . .	14
5. Import Module by YANG Semantic Version . . . . .	15
5.1. The recommended-min-version Extension . . . . .	15
5.2. Import by YANG Semantic Version Rules . . . . .	16
6. Guidelines for Using Semver During Module Development . . . . .	17
6.1. Pre-release Version Precedence . . . . .	18
6.2. YANG Semver in IETF Modules . . . . .	18
6.2.1. Guidelines for IETF Module Development . . . . .	19
6.2.2. Guidelines for Published IETF Modules . . . . .	19
7. Updates to ietf-yang-library . . . . .	19
7.1. YANG library versioning augmentations . . . . .	20
7.1.1. Advertising version . . . . .	20
8. YANG Modules . . . . .	20
9. Contributors . . . . .	26
10. Acknowledgments . . . . .	27
11. Security Considerations . . . . .	27
12. IANA Considerations . . . . .	28
12.1. YANG Module Registrations . . . . .	28
12.2. Guidance for YANG Semver in IANA maintained YANG modules and submodules . . . . .	29
13. References . . . . .	29
13.1. Normative References . . . . .	29

13.2. Informative References . . . . . 30  
Appendix A. Example IETF Module Development . . . . . 32  
Authors' Addresses . . . . . 33

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.

This document defines a YANG extension that tags a YANG artifact (i.e., YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages] ) with a version identifier that adheres to extended semantic versioning rules [SemVer]. The goal being to add a human readable version identifier that provides compatibility information for the YANG artifact without needing to compare or parse its body. The version identifier and rules defined herein represent the RECOMMENDED approach to apply versioning to IETF YANG artifacts. This document defines augmentations to ietf-yang-library to reflect the version of YANG modules within the module-set data.

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. Examples of How Versioning Is Applied To YANG Module Revisions

The following diagram illustrates how the branched revision history and the YANG Semver version extension statement could be used:

Example YANG module with branched revision history.

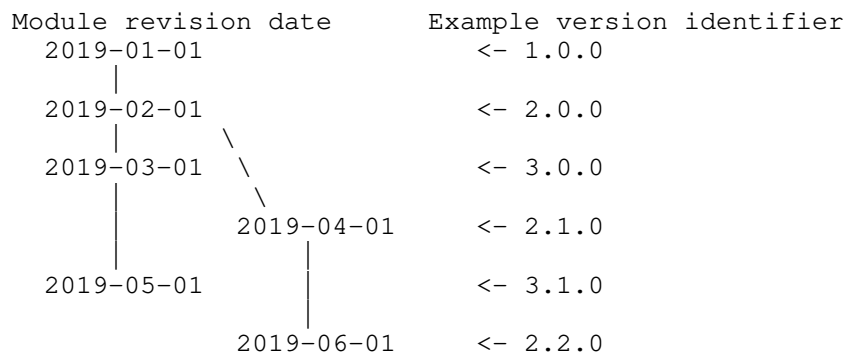


Figure 1

The tree diagram above illustrates how an example module's revision history might evolve, over time.

### 3. 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 version identifier that is consistent with the extended set of semantic versioning rules, based on [SemVer], defined within this document.

### 4. 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.

#### 4.1. 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 allows 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.

#### 4.2. YANG Semantic Version Extension

The `ietf-yang-semver` module defines a "version" extension -- a substatement to a module or submodule's "revision" statement -- that takes a YANG semantic version as its argument and specified the version for the given module or submodule. The syntax for the YANG semantic version is defined in a typedef in the same module and described below.

#### 4.3. YANG Semver Pattern

YANG artifacts that employ semantic versioning as defined in this document MUST use a version identifier 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 6. 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.

The ietf-yang-semver module included in this document defines an extension to apply a YANG Semver identifier to a YANG artifact as well as a typedef that formally specifies the syntax of the YANG Semver.

#### 4.4. Semantic Versioning Scheme for YANG Artifacts

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

- \* The YANG semantic versioning scheme is extended from version 2.0.0 of the semantic versioning scheme defined at semver.org [SemVer] to cover the additional requirements for the management of YANG artifact lifecycles that cannot be addressed using the semver.org 2.0.0 versioning scheme alone.
- \* Unlike the [SemVer] versioning scheme, the YANG semantic versioning scheme supports updates to older versions of YANG artifacts, to allow for bug fixes and enhancements to artifact versions that are not the latest. However, it does not provide for the unlimited branching and updating of older revisions which are documented by the general rules in [I-D.ietf-netmod-yang-module-versioning].
- \* YANG artifacts that use 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 identifiers 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 'ys:version' 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 identifier. 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 version identifier, then all subsequent modules in that branch (i.e., those with the same X.Y version digits) will also have a modifier. The modifier can change from "\_compatible" to "\_non\_compatible" in a subsequent version, 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 versions do not give the false impression of compatibility between two potentially non-compatible versions. If "\_non\_compatible" was removed, for example between versions "3.3.2\_non\_compatible" and "3.3.3" (where "3.3.3" was simply an editorial change), then comparing versions "3.3.3" to "3.0.0" would look like they are backwards compatible when they are not (since "3.3.2\_non\_compatible" was on the same MAJOR.MINOR branch 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.

#### 4.4.1. Branching Limitations with YANG Semver

YANG artifacts that use the YANG Semver version 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 branching approach using the following YANG Semver identifiers is not supported:



```
3.5.0 -- 3.6.0 (add leaf foo)
|
3.20.0 (added leaf bar)
```

In this case, given only the YANG Semver identifiers 3.6.0 and 3.20.0, 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 branching, where two versions on the same branch have different backwards compatible changes is allowed in [I-D.ietf-netmod-yang-module-versioning].

#### 4.4.2. 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 4.3 and Section 4.4 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 resulting schema of the including module. In this case:

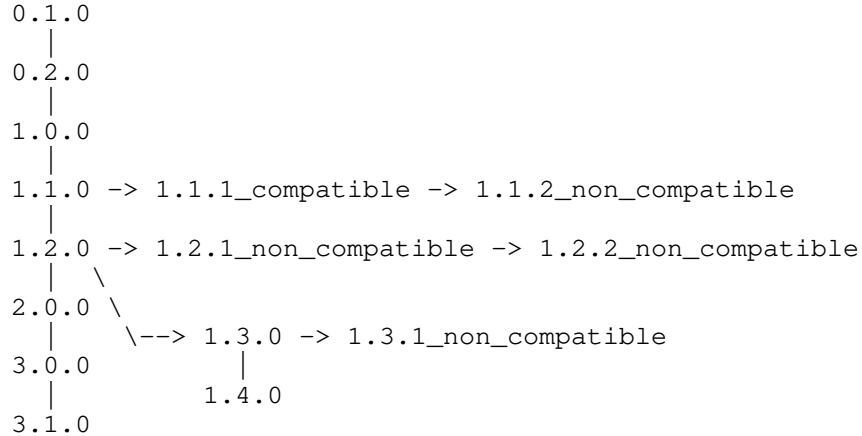
1. The including module has editorial changes
2. The submodule with the schema definition removed has non-backwards-compatible changes
3. The submodule with the schema definitions added has backwards-compatible changes

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).

#### 4.4.3. 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 branching relationships between the versions. It does not describe the chronology of the versions (i.e. when in time each version was published relative to the other versions).

The following description lists an example of what the chronological order of the versions could look like, from oldest version 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)

#### 4.5. YANG Semantic Version Update Rules

When a new version of an artifact is produced, then the following rules define how the YANG semantic version for the new artifact is calculated, based on the changes between the two artifact versions, and the YANG semantic version of the original artifact 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 identifier depends on the format of the current version identifier:
  - 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 identifiers 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 6 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 6.

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

1. An artifact author MAY update the version identifier 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 versions. That is, an artifact's version history could be 1.0.0, 1.1.0, and 1.3.0 where 1.2.0 is skipped.

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.

#### 4.6. Examples of the YANG Semver Label

##### 4.6.1. Example Module Using YANG Semver

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

```
module example-versioned-module {
  yang-version 1.1;
  namespace "urn:example:versioned:module";
  prefix "exvermod";

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

  description
    "to be completed";

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

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

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

  revision 2017-04-03 {
    description "Add new functionality, leaf 'foo'";
    ys:version 1.1.0;
  }
}
```

```
revision 2017-02-07 {
  description "First release version.";
  ys:version 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";
//   ys:version 0.2.0;
//   rev:non-backwards-compatible; // optional
//                                   // (theoretically no
//                                   // 'previous released version')
// }

// revision 2017-01-26 {
//   description "Initial module version";
//   ys:version 0.1.0;
// }

//YANG module definition starts here
}
```

#### 4.6.2. Example of Package Using YANG Semver

Below is an example YANG package that uses the YANG Semver version identifier 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 2

## 5. Import Module by YANG Semantic Version

[I-D.ietf-netmod-yang-module-versioning] allows for imports to be done based on the earliest supported date and later using the `rev:recommended-min-date` extension. This section defines a similar extension for controlling import by YANG semantic version, as well as the rules for how imports are resolved.

### 5.1. The recommended-min-version Extension

The `ietf-yang-semver` module defines a "recommended-min-version" extension -- a substatement to the "import" statement -- that takes a YANG semantic version as its argument and specifies that the minimum version of the associated module being imported SHOULD be greater than or equal to the specified value. The specific conditions for determining if a module's version is greater than or equal is defined in Section 5.2 below. Multiple `recommended-min-version` statements MAY be specified. If there are multiple `recommended-min-version` statements, they are treated as a logical OR. Removing `recommended-min-version` statements is considered a backwards compatible change. An example use is:

```
import example-module {
  ys:recommended-min-version 3.0.0;
}
```

## 5.2. Import by YANG Semantic Version Rules

A module to be imported is considered as meeting the recommended minimum version criteria if it meets one of the following conditions::

1. Has the exact MAJOR, MINOR, PATCH and "\_compatible" or "\_non\_compatible" modifiers as in the recommend-min-version value.
2. Has the same MAJOR and MINOR version numbers and a greater PATCH number. In this case, "\_compatible" and "\_non\_compatible modifiers" are ignored.
3. Has the same MAJOR version number and greater MINOR number. In this case the PATCH number and the "\_compatible" and "\_non\_compatible" modifiers are ignored.
4. Has a greater MAJOR version number. In this case MINOR and PATCH numbers and "\_compatible" and "\_non\_compatible" modifiers are ignored.

If the recommended-min-version is specified as 3.1.0, the following examples would be satisfy that recommend-min-version:

3.1.0 (by condition 1 above)

3.1.1 (by condition 2 above)

3.2.0 (by condition 3 above)

4.1.2 (by condition 4 above)

3.1.1\_compatible (by condition 2 above, noting that modifiers are ignored)

3.1.2\_non\_compatible (by condition 2 above, noting that modifiers are ignored)

If an import by recommended-min-version cannot locate a module with a version that is viable according to the conditions above, the YANG compiler SHOULD emit a warning, and then continue to resolve the import based on established [RFC7950] rules.



## 6. 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 the YANG Semver versioning 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 number. For example, an initial module or submodule version might be either 0.0.1 or 1.0.0-alpha.1. If the authors choose to use the 0 MAJOR version number scheme, they MAY switch to the pre-release scheme with a MAJOR version number of 1 when the module or submodule is nearing initial release (e.g., a module's or submodule's version 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 versioning 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 version 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 identifier (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 version 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).

### 6.1. Pre-release Version Precedence

As a module or submodule is developed, the scope of the work may change. That is, while a released module or submodule with version 1.0.0 is initially intended to become 2.0.0 in its next released 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 identifiers 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.

### 6.2. YANG Semver in IETF Modules

All published IETF modules and submodules **MUST** use YANG semantic versions in their revisions.

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 version **MUST** use the target (i.e., intended) MAJOR and MINOR version components with a 0 PATCH version number. If the intended RFC release will be non-backwards-compatible with the current RFC release, the MINOR version number **MUST** be 0.

### 6.2.1. Guidelines for IETF Module Development

All IETF modules and submodules in development MUST use the whole document name as a pre-release version identifier, 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 versions 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.

### 6.2.2. Guidelines for Published IETF Modules

For IETF YANG modules and submodules that have already been published, versions 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 4.5. 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).

## 7. 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-semver, that augments YANG library [RFC8525] with a version leaf for modules and submodules.

### 7.1. YANG library versioning augmentations

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

```

module: ietf-yang-library-semver

  augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
    +--ro version?  ys:version
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module
    /yanglib:submodule:
    +--ro version?  ys:version
  augment /yanglib:yang-library/yanglib:module-set
    /yanglib:import-only-module:
    +--ro version?  ys:version
  augment /yanglib:yang-library/yanglib:module-set
    /yanglib:import-only-module/yanglib:submodule:
    +--ro version?  ys:version

```

Figure 3

#### 7.1.1. Advertising version

The ietf-yang-library-semver YANG module augments the "module" and "submodule" lists in ietf-yang-library with "version" leafs to optionally declare the version identifier associated with each module and submodule.

## 8. YANG Modules

This YANG module contains the typedef for the YANG semantic version and the identity to signal its use.

```

<CODE BEGINS> file "ietf-yang-semver@2024-03-01.yang"
module ietf-yang-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-semver";
  prefix ys;

  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) 2024 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).

  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.

  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 ys:version to "1.0.0".

revision 2024-03-01 {
  ys:version "1.0.0-draft-ietf-netmod-yang-semver-13";
  description
    "Initial revision";
  reference
    "RFC XXXX: YANG Semantic Versioning.";
}

/*
```

```
* Extensions
*/

extension version {
  argument yang-semantic-version;
  description
    "The version extension can be used to provide an additional
    identifier associated with a module or submodule
    revision.

    The format of the version extension argument MUST conform
    to the 'version' typedef defined in this module.

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

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

    Adding a version is a backwards-compatible
    change. Changing or removing an existing version in
    the revision history is a non-backwards-compatible
    change, because it could impact any references to that
    version.";
  reference
    "XXXX: YANG Semantic Versioning;
    Section 3.2, YANG Semantic Version Extension";
}

extension recommended-min-version {
  argument yang-semantic-version;
  description
    "Recommends the versions of the module that may be imported to
    one that is greater than or equal to the specified version.

    The format of the recommended-min-version extension argument
    MUST conform to the 'version' typedef defined in this module.

    The statement MUST only be a substatement of the import
    statement. Zero, one or more 'recommended-min-version'
    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-version'
```

statements is an acceptable recommended version for import.

A particular version of an imported module adheres to an import's 'recommended-min-version' extension statement if one of the following conditions are met:

- \* Has the same MAJOR and MINOR version numbers and same or greater PATCH number.
- \* Has the same MAJOR version number and greater MINOR number. In this case the PATCH number is ignored.
- \* Has a greater MAJOR version number. In this case MINOR and PATCH numbers are ignored.

Adding, removing or updating a 'recommended-min-version' statement to an import is a backwards-compatible change.";

```
reference
  "XXXX: YANG Semantic Versioning; Section 4,
  Import Module by Semantic Version";
}

/*
 * Typedefs
 */

typedef version {
  type string {
    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 version identifier are defined in the
    reference for this typedef.";
  reference
    "RFC XXXX: YANG Semantic Versioning.";
}
}
<CODE ENDS>
```

This YANG module contains the augmentations to the ietf-yang-library.

```
<CODE BEGINS> file "ietf-yang-library-semver@2024-03-02.yang"
module ietf-yang-library-semver {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-library-semver";
  prefix yl-semver;
```

```
import ietf-yang-semver {
  prefix ys;
  reference
    "XXXX: YANG Semantic Versioning";
}
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
  and submodule level version identifiers.

  Copyright (c) 2024 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.
```



```
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 ys:version with 1.0.0 and
// remove this note.

revision 2024-03-02 {
  ys:version "1.0.0-draft-ietf-netmod-yang-semver-14";
  description
    "Initial revision";
  reference
    "XXXX: YANG Semantic Versioning";
}

// library 1.0 modules-state is not augmented with version

augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
  description
    "Add a version to module information";
  leaf version {
    type ys:version;
    description
      "The version associated with this module revision.
      The value MUST match the version value in the
      specific revision of the module loaded in this module-set.";
    reference
      "XXXX: YANG Semantic Versioning;
      Section 7.1.1, Advertising version";
  }
}

augment
  "/yanglib:yang-library/yanglib:module-set/yanglib:module/"
+ "yanglib:submodule" {
  description
    "Add a version to submodule information";
  leaf version {
    type ys:version;
    description
      "The version associated with this submodule revision.
      The value MUST match the version value in the
```

```
        specific revision of the submodule included by the module
        loaded in this module-set.";
    reference
        "XXXX: YANG Semantic Versioning;
        Section 7.1.1, Advertising version";
    }
}

augment "/yanglib:yang-library/yanglib:module-set/"
    + "yanglib:import-only-module" {
    description
        "Add a version to module information";
    leaf version {
        type ys:version;
        description
            "The version associated with this module revision.
            The value MUST match the version value in the
            specific revision of the module included in this
            module-set.";
        reference
            "XXXX: YANG Semantic Versioning;
            Section 7.1.1, Advertising version";
    }
}

augment "/yanglib:yang-library/yanglib:module-set/"
    + "yanglib:import-only-module/yanglib:submodule" {
    description
        "Add a version to submodule information";
    leaf version {
        type ys:version;
        description
            "The version associated with this submodule revision.
            The value MUST match the version 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 7.1.1, Advertising version";
    }
}
}
}
<CODE ENDS>
```

## 9. Contributors

The following people made substantial contributions to this document:

Bo Wu  
lana.wubo@huawei.com

Jan Lindblad  
jlindbla@cisco.com

#### Figure 4

### 10. 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 to 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 to 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.

### 11. 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 writeable nodes. The extensions defined are only used to document YANG artifacts.

## 12. IANA Considerations

### 12.1. YANG Module Registrations

This document requests IANA to register URIs 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-semver

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-library-semver

Registrant Contact: The IESG.

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

The following YANG modules are 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: ys

Reference: [RFCXXXX]

The ietf-yang-library-semver module:

Name: ietf-yang-library-semver

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

Prefix: yl-semver

Reference: [RFCXXXX]

## 12.2. 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 version identifier for all new revisions, as defined in Section 4.

The YANG Semver version associated with the new revision MUST follow the rules defined in Section 4.5.

Note: For IANA maintained YANG modules and submodules that have already been published, versions 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 4.5.

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.

## 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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-11, 1 March 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning-11>>.

## 13.2. 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>>.
- [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>>.
- [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](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](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](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 `ys:version`) of the module as it's being initially developed.

Version lineage for initial module development:

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

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

Continued version progression after adoption:



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

At this point, the draft is standardized 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 progression 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 standardized, 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  
Equinix  
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

NETCONF  
Internet-Draft  
Intended status: Standards Track  
Expires: December 24, 2021

C. Feng  
Q. Ma  
Huawei  
C. Xie  
China Telecom  
June 22, 2021

System Configuration Data Handling Behavior  
draft-ma-netconf-with-system-02

Abstract

This document defines an optional "system" datastore to allow clients populate the system configuration into running datastore in the device. It also defines a capability-based extension to the NETCONF protocol that helps the NETCONF client identify how system configuration are processed by the server.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 24, 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1.	Introduction . . . . .	2
1.1.	Terminology . . . . .	3
1.2.	Requirements Language . . . . .	4
2.	System Configuration Datastore . . . . .	4
2.1.	Life Cycle of the system configuration management . . . . .	4
2.2.	"Factory-Reset" RPC Impact on System Configuration Datastore . . . . .	5
3.	System Configuration data handling Basic Modes . . . . .	5
3.1.	'auto-populate' Initialization During Reboot . . . . .	6
3.2.	'auto-populate' <edit-config> Behavior towards <running> . . . . .	6
3.3.	'no-populate' <edit-config> Behavior towards <running> . . . . .	7
4.	Retrieval of System Configuration Data . . . . .	7
5.	With System Capability . . . . .	7
5.1.	Overview . . . . .	7
5.2.	Capability Identifier . . . . .	8
5.3.	Modifications to Existing Operations . . . . .	8
5.3.1.	<get-data> Operations . . . . .	8
5.3.2.	<edit-config> Operation . . . . .	8
6.	YANG Module . . . . .	8
7.	IANA Considerations . . . . .	10
8.	Security Considerations . . . . .	10
9.	Acknowledgements . . . . .	11
10.	References . . . . .	11
10.1.	Normative References . . . . .	11
10.2.	Informative References . . . . .	11
	Appendix A. Changes between Revisions . . . . .	12
	Appendix B. Open Issues tracking . . . . .	12
	Authors' Addresses . . . . .	12

## 1. Introduction

The NETCONF protocol [RFC6241] defines ways to read configuration and state data from a NETCONF server.

In some cases, a client-configured data item refers to a system generated data item (e.g., the auto-created interfaces "eth1"), which is only present in the <operational> datastore [RFC8342]. In order for it being referenced, the duplicated system configured data item needs to be retrieved from <operational> and overridden by the client.

In some other cases, a system generated configured data item is in the when/must statement, the similar operation should also be performed to make sure a successful validation, which is cumbersome.

Furthmore, when the system generated data item gets updated, there is no way to synchronize the update into <running> and the client can't detect the update automatically.

This document defines a "system" datastore to hold all the configurations provided by the system itself. It also defines a capability-based extension to the NETCONF protocol that allows the configuration synchronization between <system> and <running> both automatically and explicitly.

### 1.1. Terminology

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

The following terms are defined in this document as follows:

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

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

**physical resource independent system configuration:** When the device is powered on, the pre-provisioned configuration will be activated and provided, irrespective of physical resource present or not, sometimes the pre-provisioned configuration will be provided without must/when statement constraint (e.g., loop back interface activation), sometimes not, e.g., only provided when a special functionality is enabled.

**Physical resource dependent system configuration:** When the device is powered on and the 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.

## 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.

## 2. System Configuration Datastore

Following guidelines for defining Datastores in the appendix A of [RFC8342], this document introduces a new datastore resource named 'system' that represents the system configuration.

- o Name: "system"
- o YANG modules: all
- o YANG nodes: all "config true" data nodes
- o Management operations: The content of the datastore is set by the server in an implementation dependent manner. The content can not be changed by management operations via NETCONF, RESTCONF, the CLI, etc unless specialized, dedicated operations are provided. The datastore can be read using the standard NETCONF/RESTCONF protocol operations.
- o Origin: This document does not define any new origin identity when it interacts with <operational> datastore. The system origin Metadata Annotation is used to indicate the origin of a data item.
- o Protocols: RESTCONF, NETCONF and other management protocol.
- o Defining YANG module: "ietf-netconf-with-system".

The datastore content is usually defined by the device vendor. It is static at most of time and MAY change e.g., depending on external factors like HW available or during device upgrade. <system> does not persist across reboots. It will be automatically loaded when the device is powered on or the physical resource is present.

### 2.1. Life Cycle of the system configuration management

When the device is powered on, physical resource independent system configuration will be created in <system> automatically by the system if there is no when/must statement constraint associated with system configuration data or provided only when a special functionality is enabled.

When the device is powered on and the physical resource is inserted into the device, physical resource dependent system configuration will be automatically loaded into <system>;

When the physical resource is removed from the device, the physical resource dependent system configuration will be automatically removed from <system>;

## 2.2. "Factory-Reset" RPC Impact on System Configuration Datastore

[RFC8808] defines a "factory-reset" RPC to allow clients to reset a server back to its factory-default condition. Upon receiving the RPC, all supported conventional read-write configuration datastore (i.e., <running>, <startup> and <candidate>) are reset to the contents of <factory-default>. <system> should also immediately reset to its factory default state. That's to say, any system configurations generated due to system upgrading or client-enabled functionality should be discarded. System configuration which is generated at the first time when it boots after being shipped from factory should be retained.

## 3. System Configuration data handling Basic Modes

Not all server implementations treat system configuration data in the same way. Instead of forcing a single implementation strategy, this document allows a server advertise a particular style of system configuration data handling, and the client can adjust behavior accordingly.

This document specifies two standard system configuration handling basic modes that a server implementor may choose from:

- o auto-populate
- o no-populate

A server that uses the 'auto-populate' basic mode MUST automatically

- o Update <running> with the system configuration change, after the "system" configuration has been altered as a consequence of a plug and play operation or device powering on operation. However the configurations in <running> can not be removed automatically when configuration data nodes in <system> is deleted since those configurations in <running> are likely to have already been modified or referenced.
- o The system configuration doesn't need to be explicitly set by the client first before the system configuration needs to be updated

with client set configuration or referenced by client set configuration.

A server that uses the 'no-populate' basic mode

- o MUST not update <running> with the system configuration.
- o The system configuration MUST be explicitly set by the client first before the system configuration needs to be updated with client set configuration or referenced by client set configuration.

### 3.1. 'auto-populate' Initialization During Reboot

The contents of <system> don't have to be persist across reboots, even in the presence of non-volatile storage.

For the NETCONF server which implements the <factory-default> datastore [RFC8808], it may load <factory-default> at the first time when it boots after being shipped from factory or reset to its factory default condition. Then it's just like each normal boot time, the device generates system configurations and saves into <system>. Then the device loads the saved startup configuration (if <startup> exists) into <running>. Lastly, the device loads <system> into <running>. If there exists any conflict, the configuration in the <running> should succeed.

### 3.2. 'auto-populate' <edit-config> Behavior towards <running>

For a data node that is loaded from <system> automatically, the server MUST consider it to exist.

- o A valid 'create' operation attribute for a data node that is loaded from <system> and set by the server MUST fail with a 'data-exists' error-tag;
- o A valid 'delete' operation attribute for a data node that is loaded from <system> and set by the server MUST succeed. The deleted system configuration MUST be reloaded into <running> immediately if the system configuration is still present in the <system>;
- o A valid 'merge' operation attribute for a data node that is loaded from <system> and set by the server MUST succeed.



### 3.3. 'no-populate' <edit-config> Behavior towards <running>

The server MUST NOT consider any system configuration data node to exist in <running> configuration datastore, except those explicitly set by the client.

- o A valid 'create' operation attribute for a data node that is set by the server MUST succeed since the system configuration data is not present in the <running> configuration datastore.
- o A valid 'merge' operation attribute for a data node that is set by the server MUST succeed even though the name of data node in <system> is same as name of data node explicitly set by the client.
- o A valid 'delete' operation attribute for a data node that is set by the client MUST succeed even though the name of data node in <system> is same as name of data node explicitly set by the client. A valid 'delete' operation attribute for a data node that is not explicitly set by the client MUST fail since system configuration is not loaded into <running>.

## 4. Retrieval of System Configuration Data

TBD

## 5. With System Capability

### 5.1. Overview

The :with-system capability indicates which system-data-handling basic mode is supported by the server. These basic modes allow a NETCONF client to control whether system configuration data is returned by the server. Sending of system configuration data is controlled for each individual operation separately.

A NETCONF server implementing the :with-system capability:

- o MUST indicate its basic mode behavior by including the 'basic-mode' parameter in the capability URI;
- o MUST support the YANG module defined in Section 6 for the system configuration data handling mode indicated by the 'basic-mode' parameter.

## 5.2. Capability Identifier

```
urn:ietf:params:netconf:capability:with-system:1.0
```

The identifier MUST have a parameter: "basic-mode". This indicates how the server will treat system configuration data, as defined in Section 3. The allowed values of this parameter are 'auto-populate', and 'no-populate', as defined in Section 3.

```
urn:ietf:params:netconf:capability:with-system:1.0?basic-mode=no-populate
```

## 5.3. Modifications to Existing Operations

### 5.3.1. <get-data> Operations

As defined in Section 6, retrieval of system configuration in <system> can be used through <get-data> operation with the "datastore" parameter set to "system".

### 5.3.2. <edit-config> Operation

The <edit-config> operation has several editing modes. The 'create', and 'delete' editing operations are affected by the system configuration data handling basic mode. The other enumeration values for the NETCONF operation attribute are not affected.

If the operation attribute contains the value 'create', and the data node already exists in the target configuration datastore, then the server MUST return an <rpc-error> response with a 'invalid-value' error-tag.

If the client sets a data node that is explicitly set by the server, the server MUST accept the request if it is valid. The server MUST keep or discard the new value based on its system configuration data handling basic mode.

## 6. YANG Module

This YANG module uses the "datastore" identity [RFC8342]. Every NETCONF server which supports :with-system capability MUST implement this YANG module.

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

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

organization
  "IETF NETCONF (Network Configuration Protocol) Working Group";

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

description
  "This module defines an extension to the NETCONF protocol
  that allows the NETCONF client to control how system configuration
  data are handled by the server in particular NETCONF operations.

  Copyright (c) 2010 IETF Trust and the persons identified as
  the document authors. All rights reserved.

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

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";
// RFC Ed.: replace XXXX with actual RFC number and remove this note

revision 2021-05-14 {
  description
    "Initial version.";
  reference
    "RFC XXXX: With-system capability for NETCONF";
}

feature system-datstore {
  description
    "Indicates that the system configuration is available as a datastore.";
}

identity system {
  if-feature "system-datstore";
}
```

```
    base ds:datastore;
    description
      "This read-only datastore contains the system configuration for the
       device that will be loaded into <running> automatically in the
       "auto-populate" basic mode.";
  }
}
<CODE ENDS>
```

## 7. IANA Considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol Capability URNs registry':

urn:ietf:params:netconf:capability:with-system:1.0

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

URI: urn:ietf:params:xml:ns:netconf:system:1.0

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

Registrant Contact: The NETCONF WG of the IETF.

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

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

name: ietf-netconf-with-system

prefix: ncws

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

RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

## 8. 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.

## 9. Acknowledgements

Thanks to Robert Wilton, Kent Watsen, Balazs Lengyel, Timothy Carey for reviewing, and providing important input to, this document.

## 10. References

### 10.1. Normative References

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

### 10.2. Informative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <<https://www.rfc-editor.org/info/rfc8407>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.
- [RFC8808] Wu, Q., Lengyel, B., and Y. Niu, "A YANG Data Model for Factory Default Settings", RFC 8808, DOI 10.17487/RFC8808, August 2020, <<https://www.rfc-editor.org/info/rfc8808>>.

## Appendix A. Changes between Revisions

v01 - v02

- o Remove System configuration data retrieval behavior in the mainbody and examples in the appendix.
- o Remove <get> operation and <get-config> operation extension from the YANG data model.
- o Change basic mode values into auto-populate, no-populate.
- o Consider <factory-default> to work together with <system>.

## Appendix B. Open Issues tracking

- o Do we need to define RPC to allow the server loads <system> configuration data into <running>?
- o Can we introduce better terminology?
- o Should we define a standard operation of system configuration retrieval?

## Authors' Addresses

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

Email: frank.fengchong@huawei.com

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

Email: maqiufang1@huawei.com

Chongfeng Xie  
China Telecom  
Beijing  
China

Email: xiechf@chinatelecom.cn