Workgroup: Network Working Group
Internet-Draft:
draft-ietf-netmod-yang-module-versioning-06
Updates: 6020, 7950, 8407, 8525 (if approved)
Published: 10 July 2022
Intended Status: Standards Track
Expires: 11 January 2023
Authors: R. Wilton, Ed. R. Rahman, Ed.
Cisco Systems, Inc.
B. Lengyel, Ed. J. Clarke J. Sterne
Ericsson Cisco Systems, Inc. Nokia

Updated YANG Module Revision Handling

Abstract

This document specifies a new YANG module update procedure that can document when non-backwards-compatible changes have occurred during the evolution of a YANG module. It extends the YANG import statement with an earliest revision filter to better represent inter-module dependencies. It provides guidelines for managing the lifecycle of YANG modules and individual schema nodes. It provides a mechanism, via the revision-label YANG extension, to specify a revision identifier for YANG modules and submodules. This document updates RFC 7950, RFC 6020, RFC 8407 and RFC 8525.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction
  1.1. Updates to YANG RFCs
2. Terminology and Conventions
3. Refinements to YANG revision handling
  3.1. Updating a YANG module with a new revision
    3.1.1. Backwards-compatible rules
    3.1.2. Non-backwards-compatible changes
  3.2. non-backwards-compatible revision extension statement
  3.3. Removing revisions from the revision history
  3.4. Revision label
    3.4.1. File names
    3.4.2. Revision label scheme extension statement
  3.5. Examples for updating the YANG module revision history
4. Import by derived revision
  4.1. Module import examples
5. Updates to ietf-yang-library
  5.1. Resolving ambiguous module imports
  5.2. YANG library versioning augmentations
    5.2.1. Advertising revision-label
    5.2.2. Reporting how deprecated and obsolete nodes are handled
6. Versioning of YANG instance data
7. Guidelines for using the YANG module update rules
  7.1. Guidelines for YANG module authors
    7.1.1. Making non-backwards-compatible changes to a YANG module
  7.2. Versioning Considerations for Clients
8. Module Versioning Extension YANG Modules
9. Contributors
10. Security Considerations
11. IANA Considerations
  11.1. YANG Module Registrations
  11.2. Guidance for versioning in IANA maintained YANG modules
12. References
  12.1. Normative References
  12.2. Informative References
Appendix A. Examples of changes that are NBC
Appendix B. Examples of applying the NBC change guidelines
  B.1. Removing a data node
  B.2. Changing the type of a leaf node
  B.3. Reducing the range of a leaf node

B.4. Changing the key of a list
B.5. Renaming a node
Authors' Addresses

1. Introduction

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

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

The document comprises five parts:

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

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

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

* Considerations of how versioning applies to YANG instance data.

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

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

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

1.1. Updates to YANG RFCs

This document updates [RFC7950] section 11 and [RFC6020] section 10. Section 3 describes modifications to YANG revision handling and update rules, and Section 4 describes a YANG extension statement to do import by derived revision.

This document updates [RFC7950] section 5.2 and [RFC6020] section 5.2. Section 3.4.1 describes the use of a revision label in the name of a file containing a YANG module or submodule.

This document updates [RFC7950] section 5.6.5 and [RFC8525]. Section 5.1 defines how a client of a YANG library datastore schema resolves ambiguous imports for modules which are not "import-only".

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

This document updates [RFC8525] with augmentations to include revision labels in the YANG library data and two boolean 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.

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.

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

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

[RFC7950] section 11 and [RFC6020] section 10 require that all updates to a YANG module are BC to the previous revision of the module. This document introduces a method to indicate that an NBC change has occurred between module revisions: this is done by using a new "non-backwards-compatible" YANG extension statement in the module revision history.

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

3.1. Updating a YANG module with a new revision

This section updates [RFC7950] section 11 and [RFC6020] section 10 to refine the rules for permissible changes when a new YANG module revision is created.

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

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

As per [RFC7950] and [RFC6020], all published revisions of a module are given a new unique revision date. This applies even for module revisions containing (in the module or included submodules) only changes to any whitespace, formatting, comments or line endings (e.g., DOS vs UNIX).

3.1.1. Backwards-compatible rules

A change between two module revisions is defined as being "backwards-compatible" if the change conforms to the module update rules specified in [RFC7950] section 11 and [RFC6020] section 10, updated by the following rules:

* A "status" "deprecated" statement MAY be added, or changed from "current" to "deprecated", but adding or changing "status" to "obsolete" is not a backwards-compatible change.

* YANG schema nodes with a "status" "obsolete" substatement MAY be removed from published modules, and are classified as backwards-compatible changes. In some circumstances it may be helpful to retain the obsolete definitions since their identifiers may still be referenced by other modules and to ensure that their identifiers are not reused with a different meaning.

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

* A statement that is defined using the YANG "extension" statement MAY be added, removed, or changed, if it does not change the semantics of the module. Extension statement definitions SHOULD

specify whether adding, removing, or changing statements defined by that extension are backwards-compatible or non-backwards-compatible.

* Any changes (including whitespace or formatting changes) that do not change the semantic meaning of the module are backwards compatible.

3.1.2. Non-backwards-compatible changes

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

3.2. non-backwards-compatible revision extension statement

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

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

3.3. Removing revisions from the revision history

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

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

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

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

Example revision history:

```
revision 2020-11-11 {
  rev:revision-label 4.0.0;
  rev:non-backwards-compatible;
}

revision 2020-08-09 {
  rev:revision-label 3.0.0;
  rev:non-backwards-compatible;
}

revision 2020-06-07 {
  rev:revision-label 2.1.0;
}

revision 2020-02-10 {
  rev:revision-label 2.0.0;
  rev:non-backwards-compatible;
}

revision 2019-10-21 {
  rev:revision-label 1.1.3;
}

revision 2019-03-04 {
  rev:revision-label 1.1.2;
}

revision 2019-01-02 {
  rev:revision-label 1.1.1;
}
```

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

3.4. Revision label

Each revision entry in a module or submodule MAY have a revision label associated with it, providing an alternative alias to identify a particular revision of a module or submodule. The revision label could be used to provide an additional versioning identifier associated with the revision.

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

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

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

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

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

3.4.1. File names

This section updates [RFC7950] section 5.2 and [RFC6020] section 5.2.

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

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

   E.g., acme-router-module@2018-01-25.yang
   E.g., acme-router-module#2.0.3.yang
```

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

3.4.2. Revision label scheme extension statement

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

* specifies the revision-label scheme used by the module or submodule

* is defined in the document which specifies the revision-label scheme

* MUST be an identity derived from "revision-label-scheme-base".

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

3.5. Examples for updating the YANG module revision history

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

Example YANG module with branched revision history.

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

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

Example module, revision 2019-06-01:

```
 module example-module {

   namespace "urn:example:module";
   prefix "prefix-name";
   rev:revision-label-scheme "yangver:yang-semver";

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

   description
     "to be completed";

   revision 2019-06-01 {
     rev:revision-label 3.1.0;
     description "Add new functionality.";
   }

   revision 2019-03-01 {
     rev:revision-label 3.0.0;
     rev:non-backwards-compatible;
     description
       "Add new functionality. Remove some deprecated nodes.";
   }

   revision 2019-02-01 {
     rev:revision-label 2.0.0;
     rev:non-backwards-compatible;
     description "Apply bugfix to pattern statement";
   }

   revision 2019-01-01 {
     rev:revision-label 1.0.0;
     description "Initial revision";
   }

   //YANG module definition starts here
 }
```

Example module, revision 2019-05-01:

```
     module example-module {

       namespace "urn:example:module";
       prefix "prefix-name";
       rev:revision-label-scheme "yangver:yang-semver";

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

       description
         "to be completed";

       revision 2019-05-01 {
         rev:revision-label 2.2.0;
         description "Backwards-compatible bugfix to enhancement.";
       }

       revision 2019-04-01 {
         rev:revision-label 2.1.0;
         description "Apply enhancement to older release train.";
       }

       revision 2019-02-01 {
         rev:revision-label 2.0.0;
         rev:non-backwards-compatible;
         description "Apply bugfix to pattern statement";
       }

       revision 2019-01-01 {
         rev:revision-label 1.0.0;
         description "Initial revision";
       }

       //YANG module definition starts here
     }
```

4. Import by derived revision

[RFC7950] and [RFC6020] allow YANG module "import" statements to optionally require the imported module to have a particular revision date. In practice, importing a module with an exact revision date is often too restrictive because it requires the importing module to be updated whenever any change to the imported module occurs. The alternative choice of using an import statement without any revision date statement is also not ideal because the importing module may not work with all possible revisions of the imported module.

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

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

> The argument to the "revision-or-derived" extension statement is a revision date or a revision label.
>
> A particular revision of an imported module satisfies an import's "revision-or-derived" extension statement if the imported module's revision history contains a revision statement with a matching revision date or revision label.
>
> An "import" statement MUST NOT contain both a "revision-or-derived" extension statement and a "revision-date" statement.
>
> The "revision-or-derived" extension statement MAY be specified multiple times, allowing the import to use any module revision that satisfies at least one of the "revision-or-derived" extension statements.
>
> The "revision-or-derived" extension statement does not guarantee that all module revisions that satisfy an import statement are necessarily compatible; it only gives an indication that the revisions are more likely to be compatible. Hence, NBC changes to an imported module may also require new revisions of any importing modules, updated to accommodation those changes, along with updated import "revision-or-derived" extension statements to depend on the updated imported module revision.
>
> Adding, modifying or removing a "revision-or-derived" extension statement is considered to be a BC change.

4.1. Module import examples

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

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

4.1.1. Example 1

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

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

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

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

4.1.2. Example 2

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

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

4.1.3. Example 3

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

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

5. Updates to ietf-yang-library

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

5.1. Resolving ambiguous module imports

A YANG datastore schema, defined in [RFC8525], can specify multiple revisions of a YANG module in the schema using the "import-only" list, with the requirement from [RFC7950] section 5.6.5 that only a single revision of a YANG module may be implemented.

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

The following two rules remove the ambiguity:

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

If a module import statement could resolve to more than one module revision defined in the datastore schema, and none of those revisions are implemented, then the import MUST resolve to the module revision with the latest revision date.

5.2. YANG library versioning augmentations

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

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

5.2.1. Advertising revision-label

The ietf-yang-library-revisions YANG module augments the "module" and "submodule" lists in ietf-yang-library with "revision-label" leafs to optionally declare the revision label associated with each module and submodule.

5.2.2. Reporting how deprecated and obsolete nodes are handled

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

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

obsolete-nodes-absent: If set to "true", this leaf indicates that the server does not implement any status "obsolete" schema nodes. If this leaf is set to "false" or absent, then the behaviour is unspecified.

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

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

6. Versioning of YANG instance data

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

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

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

7. Guidelines for using the YANG module update rules

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

7.1. Guidelines for YANG module authors

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

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

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

A "rev:non-backwards-compatible" statement MUST be added if there are NBC changes relative to the previous revision.

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

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

A module that includes submodules SHOULD use the "revision-date" statement to include specific submodule revisions. The revision of the including module MUST be updated when any included submodule has changed.

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

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

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

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

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

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

* If the server can support multiple revisions of the YANG module or of YANG packages (as specified in [I-D.ietf-netmod-yang-packages]), and allows the client to select the revision (as per [I-D.ietf-netmod-yang-ver-selection]), then NBC changes MAY be done without using "status" statements. Clients would be required to select the revision which they support and the NBC change would have no impact on them.

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

1. The changes should be made gradually, e.g., a data node's status SHOULD NOT be changed directly from "current" to "obsolete" (see Section 4.7 of [RFC8407]), instead the status SHOULD first be marked "deprecated". At some point in the future, when support is removed for the data node, there are two options. The first, and preferred, option is to keep the data node definition in the model and change the status to "obsolete". The second option is to simply remove the data node from the model, but this has the risk of breaking modules which import the modified module, and the removed identifier may be accidently reused in a future revision.

2. For deprecated data nodes the "description" statement SHOULD also indicate until when support for the node is guaranteed (if known). If there is a replacement data node, rpc, action or notification for the deprecated node, this SHOULD be stated in the "description". The reason for deprecating the node can also be included in the "description" if it is deemed to be of potential interest to the user.

3. For obsolete data nodes, it is RECOMMENDED to keep the above information, from when the node had status "deprecated", which is still relevant.

4. When obsoleting or deprecating data nodes, the "deprecated" or "obsolete" status SHOULD be applied at the highest possible level in the data tree. For clarity, the "status" statement SHOULD also be applied to all descendent data nodes, but the additional status related information does not need to be repeated if it does not introduce any additional information.

5. NBC changes which can break imports SHOULD be avoided because of the impact on the importing module. The importing modules could get broken, e.g., if an augmented node in the importing module has been removed from the imported module. Alternatively, the schema of the importing modules could undergo an NBC change due to the NBC change in the imported module, e.g., if a node in a grouping has been removed. As described in Appendix B.1, instead of removing a node, that node SHOULD first be deprecated and then obsoleted.

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

7.2. Versioning Considerations for Clients

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

* Clients SHOULD be liberal when processing data received from a server. For example, the server may have increased the range of an operational node causing the client to receive a value which is outside the range of the YANG model revision it was coded against.

* Clients SHOULD monitor changes to published YANG modules through their revision history, and use appropriate tooling to understand the specific changes between module revision. In particular, clients SHOULD NOT migrate to NBC revisions of a module without understanding any potential impact of the specific NBC changes.

* Clients SHOULD plan to make changes to match published status changes. When a node's status changes from "current" to "deprecated", clients SHOULD plan to stop using that node in a timely fashion. When a node's status changes to "obsolete", clients MUST stop using that node.

8. Module Versioning Extension YANG Modules

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

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

module ietf-yang-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-revisions";
  prefix rev;

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

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

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

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

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

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

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

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

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

    This version of this YANG module is part of RFC XXXX; see
```

```
       the RFC itself for full legal notices.

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

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

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

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

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

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

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

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

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

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

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

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

  reference
    "XXXX: Updated YANG Module Revision Handling;
    Section 3.2, non-backwards-compatible revision extension statement";
}

extension revision-label {
  argument revision-label;
  description
    "The revision label can be used to provide an additional
    versioning identifier associated with a module or submodule
    revision.  One such scheme that
    could be used is [XXXX: ietf-netmod-yang-semver].

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

    The statement MUST only be a substatement of the revision
    statement.  Zero or one revision-label statements per parent
```

```
      statement are allowed.  No substatements for this extension
      have been standardized.

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

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

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

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

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

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

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

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

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

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

```
         revision-date or revision-label.

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

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

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

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

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

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

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

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

     identity revision-label-scheme-base {
       description
         "Base identity from which all revision label schemes are
         derived.";

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

     }
   }
```

<CODE ENDS>

   YANG module with augmentations to YANG Library to revision labels

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

module ietf-yang-library-revisions {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-library-revisions";
  prefix yl-rev;

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

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

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

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

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

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

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

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

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

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

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

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

  // RFC Ed.: update the date below with the date of RFC publication
  // and remove this note.
  // RFC Ed.: replace XXXX (including in the imports above) with
  // actual RFC number and remove this note.
  // RFC Ed.: please replace revision-label version with 1.0.0 and
  // remove this note.
  revision 2021-11-04 {
    rev:revision-label 1.0.0-draft-ietf-netmod-yang-module-versioning-05;
    description
      "Initial revision";
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }

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

  augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
    description
      "Add a revision label to module information";
    leaf revision-label {
      type rev:revision-label;
      description
        "The revision label associated with this module revision.
         The label MUST match the rev:revision-label value in the specific
         revision of the module loaded in this module-set.";

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

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

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

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

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

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

      reference
        "XXXX: Updated YANG Module Revision Handling;
         Section 5.2.1, Advertising revision-label";
    }
  }
  augment "/yanglib:yang-library/yanglib:schema" {
    description
      "Augmentations to the ietf-yang-library module to indicate how
       deprecated and obsoleted nodes are handled for each datastore
       schema supported by the server.";

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

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

    leaf obsolete-nodes-absent {
      type boolean;
      description
        "If set to true, this leaf indicates that the server does not
         implement any status 'obsolete' schema nodes.  If this leaf is
         absent or set to false, then the behaviour is unspecified.";

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

<CODE ENDS>
```

9. Contributors

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

*Balazs Lengyel

* Benoit Claise

* Bo Wu

* Ebben Aries

* Jan Lindblad

* Jason Sterne

* Joe Clarke

* Juergen Schoenwaelder

* Mahesh Jethanandani

* Michael (Wangzitao)

* Qin Wu

* Reshad Rahman

* Rob Wilton

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

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

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

10. Security Considerations

The document does not define any new protocol or data model. There are no security considerations beyond those specified in [RFC7950] and [RFC6020].

11. IANA Considerations

11.1. YANG Module Registrations

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

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

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

The ietf-yang-revisions module:

Name: ietf-yang-revisions

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

Prefix: rev

Reference: [RFCXXXX]

The ietf-yang-library-revisions module:

Name: ietf-yang-library-revisions

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

Prefix: yl-rev

Reference: [RFCXXXX]

11.2. Guidance for versioning in IANA maintained YANG modules

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

IANA is responsible for maintaining and versioning YANG modules that are derived from other IANA registries. For example, "iana-if-type.yang" [IfTypeYang] is derived from the "Interface Types (ifType) IANA registry" [IfTypesReg], and "iana-routing-types.yang"

[RoutingTypesYang] is derived from the "Address Family Numbers" [AddrFamilyReg] and "Subsequent Address Family Identifiers (SAFI) Parameters" [SAFIReg] IANA registries.

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

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

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

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

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

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

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

12. References

12.1. Normative References

[I-D.ietf-netmod-rfc6991-bis]
Schoenwaelder, J., "Common YANG Data Types", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc6991-bis-13, 22 March 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-rfc6991-bis-13.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>.

[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>.

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

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

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

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

12.2. Informative References

[AddrFamilyReg]
"Address Family Numbers IANA Registry", <https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>.

[I-D.clacla-netmod-yang-model-update] Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://www.ietf.org/archive/id/draft-clacla-netmod-yang-model-update-06.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-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://www.ietf.org/archive/id/draft-ietf-netmod-yang-packages-03.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>.

[IfTypesReg] "Interface Types (ifType) IANA Registry", <https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>.

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

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

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

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

[SAFIReg] "Subsequent Address Family Identifiers (SAFI) Parameters IANA Registry", <https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

Appendix A. Examples of changes that are NBC

Examples of NBC changes include:

*Deleting a data node, or changing it to status obsolete.

*Changing the name, type, or units of a data node.

*Modifying the description in a way that changes the semantic meaning of the data node.

*Any changes that change or reduce the allowed value set of the data node, either through changes in the type definition, or the addition or changes to "must" statements, or changes in the description.

*Adding or modifying "when" statements that reduce when the data node is available in the schema.

*Making the statement conditional on if-feature.

Appendix B. Examples of applying the NBC change guidelines

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

The examples are all for "config true" nodes.

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

B.1. Removing a data node

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

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

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

B.2. Changing the type of a leaf node

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

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

2. A new schema node, e.g., "vpn-name", of type string is added to the same location as the existing node "vpn-id". This new node has status "current" and its description explains that it is replacing node "vpn-id".

3. During the period of time when both schema nodes are supported, the interactions between the two nodes is outside the scope of this document and will vary on a case by case basis. Here are some options:

   1. A server may prevent the new node from being set if the old node is already set (and vice-versa). A "choice" construction could be used, or the new node may have a

"when" statement to achieve this. The old node must not have a "when" statement since this would be an NBC change, but the server could reject the old node from being set if the new node is already set.

2. If the new node is set and a client does a get or get-config operation on the old node, the server could map the value. For example, if the new node "vpn-name" has value "123" then the server could return integer value 123 for the old node "vpn-id". However, if the value can not be mapped then the configuration would be incomplete. The behavior in this case is outside the scope of this document.

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

B.3. Reducing the range of a leaf node

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

1. If all values which are being removed were never supported, e.g., if a vpn-id of 2001 or higher was never accepted, this is a BC change for the functionality (no functionality change). Even if it is an NBC change for the YANG model, there should be no impact for clients using that YANG model.

2. If one or more values being removed was previously supported, e.g., if a vpn-id of 3333 was accepted previously, this is an NBC change for the YANG model. Clients using the old YANG model will be impacted, so a change of this nature should be done carefully, e.g., by using the steps described in Appendix B.2

B.4. Changing the key of a list

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

1. The status of list "sessions" is changed to "deprecated" and the list is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate the new list that is replacing this list.

2. A new list is created in the same location with the same descendant schema nodes but with "dest-address" as key. Finding

an appropriate name for the new list can be difficult. In this case the new list is called "sessions-address", has status "current" and its description should explain that it is replacing list "session".

3. During the period of time when both lists are supported, the interactions between the two lists is outside the scope of this document and will vary on a case by case basis. Here are some options:

   1. A server could prevent entries in the new list from being created if the old list already has entries (and vice-versa).

   2. If the new list has entries created and a client does a get or get-config operation on the old list, the server could map the entries. However, if the new list has entries which would lead to duplicate keys in the old list, the mapping can not be done.

4. When list "sessions" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

If the server can support NBC revisions of the YANG module simultaneously using version selection [I-D.ietf-netmod-yang-ver-selection], then the changes can be done immediately:

1. The new revision of the YANG module has the list "sessions" modified to have "dest-address" as key, this is an NBC change.

2. Clients which require the previous functionality select the older module revision

B.5. Renaming a node

A leaf or container schema node may be renamed, either due to a spelling error in the previous name or because of a better name. For example a node "ip-adress" could be renamed to "ip-address":

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

2. The new schema node "ip-address" is added to the same location as the existing node "ip-adress". This new node has status "current" and its description should explain that it is replacing node "ip-adress".

3. During the period of time when both nodes are available, the interactions between the two nodes is outside the scope of this document and will vary on a case by case basis. Here are some options:

    1. A server may prevent the new node from being set if the old node is already set (and vice-versa). A "choice" construction could be used, or the new node may have a "when" statement to achieve this. The old node must not have a "when" statement since this would be an NBC change, but the server could reject the old node from being set if the new node is already set.

    2. If the new node is set and a client does a get or get-config operation on the old node, the server could use the value of the new node. For example, if the new node "ip-address" has value X then the server may return value X for the old node "ip-adress".

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

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Reshad Rahman (editor)

Email: reshad@yahoo.com

Balazs Lengyel (editor)
Ericsson

Email: balazs.lengyel@ericsson.com

Joe Clarke
Cisco Systems, Inc.

Email: jclarke@cisco.com

Jason Sterne
Nokia

Email: jason.sterne@nokia.com