Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: November 25, 2018

B. Claise
J. Clarke
Cisco Systems, Inc.
B. Lengyel
Ericsson
K. D'Souza
AT&T

May 24, 2018

New YANG Module Update Procedure
draft-clacla-netmod-yang-model-update-05

Abstract

This document specifies a new YANG module update procedure in case of backward-incompatible changes, as an alternative proposal to the YANG 1.1 specifications. This document updates RFC 7950.

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 November 25, 2018.

Table of Contents

1. Introduction

The YANG data modeling language [RFC7950] specifies strict rules for updating YANG modules (see section 11 "Updating a Module"). Citing a few of the relevant rules:

1. "As experience is gained with a module, it may be desirable to revise that module. However, changes to published modules are not allowed if they have any potential to cause interoperability problems between a client using an original specification and a server using an updated specification."

2. "Note that definitions contained in a module are available to be imported by any other module and are referenced in "import" statements via the module name. Thus, a module name MUST NOT be changed. Furthermore, the "namespace" statement MUST NOT be changed, since all XML elements are qualified by the namespace."

3. "Otherwise, if the semantics of any previous definition are changed (i.e., if a non-editorial change is made to any definition other than those specifically allowed above), then this MUST be achieved by a new definition with a new identifier."

4. "deprecated indicates an obsolete definition, but it permits new/ continued implementation in order to foster interoperability with older/existing implementations."

What are the consequences?

1. If a YANG module is intended to update another YANG module, the module name should not be changed as it will break existing tooling and code by changing imports statements, service composition at the orchestration layer, general network management applications, etc.

2. When the same YANG module name is kept, its new revision must be updated in a backward-compatible way.

3. While most of the non-backward compatible changes are prohibited, a client still does not know if a changed module is backward compatible, as a server may remove parts of a module after marking it deprecated or obsolete.

This document specifies a new YANG module update procedure in case of backward-incompatible changes, as an alternative proposal to the YANG 1.1 specifications. This document updates RFC 7950.

This document does not address the potential need of an automatic way to discover that a YANG-MODULE-B obsoletes YANG-MODULE-A, so that YANG-MODULE-A should not be given any attention. This problem is currently solved by RFC obsolete tag as a level of indirection between the YANG modules.

2. The Problems

This section lists a series of problems, which leads to the solution in the next section.

2.1. Slow Standardization

The points made in the introduction lead to the logical conclusion that the standardized YANG modules have to be perfect on day one (at least the structure and meaning), which in turn might explain why IETF YANG modules take so long to standardize. Shooting for perfection is obviously a noble goal, but if the perfect standard comes too late, it doesn't help the industry.

2.2. Some YANG Modules Are Not Backward Compatible

As we learn from our mistakes, we're going to face more and more backward-incompatible YANG modules. An example is the YANG data model for L3VPN service delivery [RFC8049], which, based on implementation experience, has been updated in a backward-incompatible way by [RFC8299].

While Standards Development Organization (SDO) YANG modules are obviously better for the industry, we must recognize that many YANG modules are actually generated YANG modules (for example, from internal databases), also known as native YANG modules, or vendor modules [RFC8199]. From time to time, the new YANG modules are not backward-compatible.

In such cases, it would be better to indicate how backward-compatible a given YANG module actually is.

2.3. Non-Backward Compatible Errors

Sometimes small errors force us to make non-backward compatible updates. As an example imagine that we have a string with a complex pattern (e.g., an IP address). Let's assume the initial pattern incorrectly allows IP addresses to start with 355. In the next version this is corrected to disallow addresses starting with 355. Formally this is an non-backward compatible change as the value space of the string is decreased. In reality an IP address and the implementation behind it was never capable of handling an address starting with 355. So practically this is a backward compatible change, just like a correction of the description statement. Still current YANG rules would force a module name change.

2.4. YANG Module Transition Strategy

Let's assume for a moment that we change the name of a YANG module when making a backwards-incompatible change, with the specific example of ietf-routing, which some propose to update to ietf-routing-2. [yangcatalog] provides tooling that shows the interdependencies of YANG modules.

Here are the over 30 modules that depend on ietf-routing <https://www.yangcatalog.org/yang-search/impact_analysis.php?modules[]=ietf-routing&recurse=0&rfcs=1&show_subm=1&show_dir=dependents>.

Let's look at the difference for ietf-routing-2: <https://www.yangcatalog.org/yang-search/impact_analysis.php?modules[]=ietf-routing-2&recurse=0&rfcs=1&show_subm=1&show_dir=dependents>.

Changing the module name from ietf-routing to ietf-routing-2 implies that the we have to warn all draft authors of ietf-routing YANG dependent modules. First, to make sure they are aware of ietf-routing-2 (publishing a RFC8022bis mentioning in the module description that this module is not compatible with the NMDA architecture, and providing a pointer to ietf-routing-2 requires manual, tedius work). And second, to ask them to change their import (or service composition) to ietf-routing-2. Hopefully, in the ietf-routing case, most dependent YANG modules are part of the IETF, so the communication is a manageable. For the already existing dependent vendor modules the problem is worse. And then there are network management applications that may already be using ietf-routing that would require new code to handle ietf-routing-2.

Changing the ietf-interfaces YANG module name would be a different challenge, as it's used throughout the industry: <https://www.yangcatalog.org/yang-search/impact_analysis.php?modules[]=ietf-interfaces&recurse=0&rfcs=1&show_subm=1&show_dir=dependents>

2.5. Need to Allow Non-Backward Compatible changes

As described in the previous sections, there is a need to allow non-backward compatible changes without changing a module's name. This would avoid many of the above problems. Allowing non-backward compatible changes to happen without a module name change will decrease the number of separate modules to handle and will make it a trivial task to track these non-backward compatible changes.

2.6. Clear Indication of Node Support

The current definition of deprecated and obsolete in [RFC7950] (as quoted below) is problematic and should be corrected.

- "deprecated" indicates an obsolete definition, but it permits new/continued implementation in order to foster interoperability with older/existing implementations.

- "obsolete" means that the definition is obsolete and SHOULD NOT be implemented and/or can be removed from implementations.

YANG is considered an interface contract between the server and the client. The current definitions of deprecated and obsolete mean that a schema node that is either deprecated or obsolete may or may not be implemented. The client has no way to find out which is the case except for by trying to write or read data at the leaf in question. This probing would need to be done for each separate data-node, which is not a trivial thing to do. This "may or may not" is unacceptable in a contract. In effect, this works as if there would be an if-feature statement on each deprecated schema node where the server does not advertise whether the feature is supported or not. Why is it not advertised?

If a schema part is considered old/bad we need to be able to give advance warning that it will be removed. As this is an advance warning the part shall still be present and usable in the current revision; however, it will be removed in one of the next revisions. This is compounded by the fact that obsolete nodes may return bad or incorrect data. A client might expect they work by the fact they return something at all. There must be a clear indication from the server whether or not deprecated and obsolete nodes are implemented as defined.

2.7. No way to easily decide whether a change is Backward Compatible

A management system, SDN controller or any other user of a module should be capable of easily determining the compatibility between two module versions. Higher level logic for a network function, something that can not be implemented in a purely model driven way, is always dependent on a specific version of the module. If the client finds that the module has been updated on the network node, it has to decide if it tries to handle it as it handled the previous version of the model or if it just stops, to avoid problems. To make this decision the client needs to know if the module was updated in a backward compatible way or not.

This is not possible to decide today because of the following:

- It is possible to change the semantic behavior of a data node, action or rpc while the YANG definition does not change (with the possible exception of the description statement). In such a case it is impossible to determine whether the change is backward compatible just by looking at the YANG statements. Its only the human model designer that can decide.

- Problems with the deprecated and obsolete status statement, Section 2.6

- Modelers might decide to violate YANG 1.1 update rules for some of the reasons above

Finding status changes or violations of update rules need a line by line comparision of the old and new modules, no easy task.

3. The Solution

The solution is composed of five parts:

1. A semantic versioning YANG extension, along with an optional additional check that validates the semantic versioning from a syntactic point of view, which can either assist in determining the correct semantic versioning values, or which can help in determining the values for YANG modules that don't support this extension.

2. The import by version statement"

3. Updates to the YANG 1.1 module update rules

4. Updates to ietf-yang-library

5. The deprecated and obsolotes Reason"

3.1. Semantic Versioning

3.1.1. Semantic Versioning, As Set by the YANG Module Designer

The semantic versioning solution proposed here has already been proposed in [I-D.openconfig-netmod-model-catalog] (included here with the authors' permission) which itself is based on [openconfigsemver]. The goal is to indicate the YANG module backwards (in)compatibility, following semver.org semantic versioning [semver]:

"The SEMVER version number for the module is introduced. This is expressed as a semantic version number of the form: x.y.z

- x is the MAJOR version. It is incremented when the new version of the specification is incompatible with previous versions.

- y is the MINOR version. It is incremented when new functionality is added in a manner that is backward-compatible with previous versions.

o z is the PATCH version. It is incremented when bug fixes are made in a backward-compatible manner."

The semantic version value is set by the YANG module developer at the design and implementation times. Along these lines, we propose the following YANG 1.1 extension for a more generic semantic version. The formal definition is found at the end of this document.

```
extension module-version {
    argument semver;
}
```

The extension would typically be used this way:

```
module yang-module-name {

  namespace "name-space";
  prefix "prefix-name";

  import ietf-semver { prefix "semver"; }

  description
    "to be completed";


  revision 2017-10-30 {
    description
      "Change the module structure";
    semver:module-version "2.0.0";
  }

  revision 2017-07-30 {
    description
      "Added new feature XXX";
    semver:module-version "1.2.0";
  }

  revision 2017-04-03 {
    description
      "Update copyright notice.";
    semver:module-version "1.0.1";
  }

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

  revision 2017-01-26 {
    description
      "Initial module for inet types";
    semver:module-version "0.1.0";
  }


  //YANG module definition starts here
```

See also "Semantic Versioning and Structure for IETF Specifications" [I-D.claise-semver] for a mechanism to combine the semantic versioning, the GitHub tools, and a potential change to the IETF process.

3.1.2. The Derived Semantic Version

If an explicitly defined semantic version is not available in the YANG module, it is possible to algoritmically calculate a derived semantic version. This can be used for modules not containing a definitive semantic-version as defined in this document or as a starting value when specifying the definitive semantic-version. Be aware that this algorithm may sometimes incorrectly classify changes between the categories non-compatible, compatible or error-correction.

3.1.3. Implementation Experience

[yangcatalog] uses the pyang utility to calculate the derived-semantic-version for all of the modules contained within the catalog. [yangcatalog] contains many revisions of the same module in order to provide its derived-semantic-version for module consumers to know what has changed between revisions of the same module.

Two distinct leafs in the YANG module [I-D.clacla-netmod-model-catalog] contain this semver notation:

- the semantic-version leaf contains the value embedded within a YANG module (if it is available).

- the derived-semantic-version leaf is established by examining the the YANG module themselves. As such derived-semantic-version only takes syntax into account as opposed to the meaning of various elements when it computes the semantic version.

- The algorithm used to produce the derived-semantic-version is as follows:

  1. Order all modules of the same name by revision from oldest to newest. Include module revisions that are not available, but which are defined in the revision statements in one of the available module versions.

  2. If module A, revision N+1 has failed compilation, bump its derived semantic MAJOR version. For unavailable module versions assume non-backward compatible changes were done., thus bump its derived semantic MAJOR version.

  3. Else, run "pyang --check-update-from" on module A, revision N and revision N+1 to see if backward-incompatible changes exist.

4. If backward-incompatible changes exist, bump module A, revision N+1's derived MAJOR semantic version.

5. If no backward-incompatible changes exist, compare the pyang trees of module A, revision N and revision N+1.

6. If there are structural differences (e.g., new nodes), bump module A, revision N+1's derived MINOR semantic version.

7. If no structural differences exist, bump module A, revision N+1's derived PATCH semantic version.

The pyang utility checks many of the points listed in section 11 of [RFC7950] for known module incompatibilities. While this approach is a good way to programmatically obtain a semantic version number, it does not address all cases whereby a major version number might need to be increased. For example, a node may have the same name and same type, but its meaning may change from one revision of a module to another. This represents a semantic change that breaks backwards compatibility, but the above algorithm would not find it. Therefore, additional, sometimes manual, rigor must be done to ensure a proper version is chosen for a given module revision.

3.2. Import by Semantic Version

If a module is imported by another one, it is usually not specified which revision of the imported module should be used. However, not all revisions may be acceptable. Today YANG 1.1 allows one to specify the revision date of the imported module, but that is too specific, as even a small spelling correction of the imported module results in a change to its revision date, thus making the module revision ineligible for import.

Using semantic versioning to indicate the acceptable imported module versions is much more flexible. For example:

- Only a module of a specific MAJOR version is acceptable. All MINOR and PATCH versions can also be imported.

- A module at a specific MAJOR version or higher is acceptable.

- A module at a specific MAJOR.MINOR version is acceptable. All PATCH versions can also be imported.

- A module within a certain range of versions are acceptable. For example, in this case, a module between version 1.0.0 (inclusive) and 3.0.0 (exclusive) are acceptable.

The ietf-semver module provides another extension, import-versions that is a child of import and specifies the rules for an acceptable set of versions of the given module. The structure of this extension is specified as follows:

TODO: How to specify this? One thought is below, not fully formalized as this should be discussed further. Note: while this uses a comma to separate discrete versions, we could instead allow for this to be specified multiple times.

[\[(]X[.Y[.Z]][-[X[.Y[.X]]][\])]][,...]

Where the first character MAY be a '[' or '(' to indicate at least inclusive and at least
exclusive (respectively). If this is omitted, a full semantic version must be specified
and the import will only support this one version.

The following version, if specified with a '[' or '(' indicates the lower bound. This can
be a full semantic version or a MAJOR only or MAJOR.MINOR only.

The '-', if specified, is a literal hyphen indicating a range will be specified. If the second portion
of the import-versions clause is omitted, then there is no upper bound on what will be considered
an acceptable imported version.

After the '-' the upper bound semantic version (or part thereof) follows.

After the upper bound version, one of ']' or ')' MUST follow to indicate whether this limit is inclusive
or exclusive of the upper bound respectively.

Finally, a literal comma (',') MAY be specified with additional ranges. Each range is taken as a logical
OR.

For example:

```
import example-module {
  semver:import-versions "[1.0.0-3.0.0)";
  // All versions between 1.0.0 (inclusive) and 3.0.0 (exclusive) are
acceptable.
}

import example-module {
  semver:import-versions "[2-5]";
  // All versions between 2.0.0 (inclusive) and 5.y.z (inclusive) where y and z
are
  // any value for MINOR and PATCH versions.
}

import example-module {
  semver:import-versions "[1.5-2.0.0),[2.5";
  // All versions between 1.5.0 (inclusive) and 2.0.0 (exclusive) as well as
all versions
  // greater than 2.5 (inclusive).  In this manner, if 2.0 was branched from
1.4, and a
  // new feature was added into 1.5, all versions of 1.x.x starting at 1.5 are
allowed,
  // but the feature was not merged into 2.y.z until 2.5.0.
}

import example-module {
  semver:import-versions "[1";
  // All versions greater than MAJOR version 1 are acceptable.  This includes
any
  // MINOR or PATCH versions.
}

import example-module {
  semver:import-versions "1.0.0";
  // Only version 1.0.0 is acceptable (this mimics what exists with import by
revision).
}

import example-module {
  semver:import-versions "[1.1-2)"";
  // All versions greater than 1.1 (inclusive, and including all PATCH versions
off of 1.1)
  // up to MAJOR version 2 (exclusive) are acceptable.
}

import example-module {
  semver:import-versions "[1.1-2),[3";
  // All versions greater than 1.1 (inclusive, and including all PATCH versions
off of 1.1)
```

```
  // up to MAJOR version 2 (exclusive), as well as all versions greater than
MAJOR version 3
  // (inclusive) are acceptable.
}

import example-module {
  semver:import-versions "[1.1-2),[3.0.0";
  // This is equivalent to the example above, simply indicating that a partial
semantic version
  // assumes all missing components are 0.
}
```

The import statement SHOULD include a semver:import-versions statement and MUST NOT include a revision statement. An import statement MUST NOT contain both a semver:import-versions and a revision substatement. The use of the revision substatement for import should be discouraged.

3.3. Updates to YANG 1.1 Module Update Rules

RFC 7950 section 11, must be updated to allow for non-backwards changes provided they follow the semantic versioning guidelines and increase the MAJOR version number when a backwards incompatible change is made. The following is proposed text for this change.

"As experience is gained with a module, it may be desirable to revise that module. Changes to published modules are allowed, even if they have some potential to cause interoperability problems, if the module-version YANG extension is used in the revision statement to clearly indicate the nature of the change."

3.4. Updates to ietf-yang-library

The ietf-semver YANG module also specifies additional ietf-yang-library [RFC7895] [I-D.ietf-netconf-rfc7895bis] leafs to be added at the module and submodule levels. The first is module-version, which augments /yanglib:yang-library/yanglib:module-set/yanglib:module. This specifies the current semantic version of the associated module and revision in a given module-set. The related submodule-version leaf is added at /yanglib:yang-library/yanglib:module-set/yanglib:module/yanglib:submodule to indicate the semantic version of a submodule.

In order to satisfy the requirement that deprecated and obsolete node presence and operation are easily and clearly known to clients, ietf-semver also augments the ietf-yang-library with two additional boolean leafs at /yanglib:yang-library/yanglib:module-set/yanglib:module. A client can make one request of the ietf-yang-library and know whether or a not a module that has deprecated or obsolete has those nodes implemented by the server, as opposed to making multiple requests for each node in question.

deprecated-nodes-present : A boolean that indicates whether or not this server implements deprecated nodes. The value of this leaf SHOULD be true; and if so, the server MUST implement nodes within this module as they are documented. If specific deprecated nodes are not implemented as document, then they MUST be listed as deviations. This leaf defaults to true.

obsolete-nodes-present : A boolean that indicates whether or not this server implements obsolete nodes. The value of this leaf SHOULD be false; and if so, the server MUST NOT implement nodes within this module. If this leaf is true, then all nodes in this module MUST be implemented as documented in the module. Any variation of this MUST be listed as deviations. This leaf defaults to false.

If a module does not have any deprecated or obsolete nodes, the server SHOULD set the corresponding leaf above to true. This is helpful to clients, such that if the MAJOR version number has not changed, and these booleans are true, then a client does not have to check the status of any node for the module.

Module compatibility can be affected if values other than the default are used for the leafs described here. For example, if a server does not implemennt deprecated nodes, then a given module revision may be incompatible with a previous revision where the nodes were not deprecated. When calculating backwards compatibility, the default values of these leafs MUST be considered. From a client's point of view, if two module revisions have the same MAJOR version but the run-time value of deprecated-nodes-present (as read from the ietf-yang-library) is false, then compatibility MUST NOT be assumed based on the module-version alone.

3.5. Deprecated and Obsolete Reasons

The ietf-semver module specifies an extension, status-description, that is designed to be used as a substatement of the status statement when the status is deprecated or obsolete. This argument to this extension is freeform text that explains why the node was deprecated or made obsolete. It may also point to other schema elements that take the place of the deprecated or obsolete node. This text is designed for human consumption to aid in the migration away from nodes that will one day no longer work. An example is shown below.

```
leaf imperial-temperature {
  type int64;
  units "degrees Fahrenheit";
  status deprecated {
    semver:status-description
      "Imperial measurements are being phased out in favor
       of their metric equivalents.  Use metric-temperature
       instead.";
  }
  description
    "Temperature in degrees Fahrenheit.";
}
```

4. Semantic Version Extension YANG Module

The extension and related ietf-yang-library changes described in this module are defined in the YANG module below.

```
<CODE BEGINS> file "ietf-semver@2018-04-05.yang"
  module ietf-semver {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-semver";
    prefix semver;

    import ietf-yang-library {
      prefix yanglib;
    }

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

       Author:   Benoit Claise
                 <mailto:bclaise@cisco.com>

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

       Author:   Kevin D'Souza
                 <mailto:kd6913@att.com>

       Author:   Balazs Lengyel
                 <mailto:balazs.lengyel@ericsson.com>";
    description
      "This module contains a definition for a YANG 1.1 extension to
       express the semantic version of YANG modules.";

    revision 2018-04-05 {
      description
        "* Properly import ietf-yang-library.
         * Fix the name of module-semver => module-version.
         * Fix regular expression syntax.
         * Augment yang-library with booleans as to whether or not
           deprecated and obsolete nodes are present.
         * Add an extension to enable import by semantic version.
         * Add an extension status-description to track deprecated
           and obsolete reasons.
         * Fix yang-library augments to use 7895bis.";
      reference
```

```
        "draft-clacla-netmod-yang-model-update:
         New YANG Module Update Procedure";
      semver:module-version "0.2.1";
    }
    revision 2017-12-15 {
      description
        "Initial revision.";
      reference
        "draft-clacla-netmod-yang-model-update:
         New YANG Module Update Procedure";
      semver:module-version "0.1.1";
    }

    extension module-version {
      argument semver;
      description
        "The version number for the module revision it is used in.
         This is expressed as a semantic version string in the form:
          x.y.z
         where:
          * x corresponds to the major version,
          * y corresponds to a minor version,
          * z corresponds to a patch version.

         A major version number of 0 indicates that this model is still
         in development, and is potentially subject to change.

         Following a release of major version 1, all modules will
         increment major revision number where backwards incompatible
         changes to the model are made.

         The minor version is changed when features are added to the
         model that do not impact current clients use of the model.
         When major version is stepped, the minor version is reset to 0.

         The patch-level version is incremented when non-feature changes
         (such as bugfixes or clarifications to human-readable
         descriptions that do not impact model functionality) are made
         that maintain backwards compatibility.
         When major or minor version is stepped, the patch-level is
         reset to 0.

         By comparing the module-version between two revisions of a
         given module, one can know if different revisions are backwards
         compatible or not, as well as
         whether or not new features have been added to a newer revision.

         If a module contains this extension it indicates that for this
```

```
         module the updated status and update rules as this described in
         RFC XXXX are used.

         The statement MUST only be a substatement of the revision statement.
         Zero or one module-version statement is allowed per parent
         statement. NO substatements are allowed.
        ";
      reference "http://semver.org/ : Semantic Versioning 2.0.0";
    }

    extension import-versions {
      argument version-clause;
      description
        "This extension specifies an acceptable set of semantic versions of a
given module
         that may be imported.  The version-clause argument is specified in the
following
         format

         [\\[(]X[.Y[.Z]][-[X[.Y[.X]]][\\])]][,...]

         Where the first character MAY be a '[' or '(' to indicate at least
inclusive and at least
          exclusive (respectively).  If this is omitted, a full semantic
version must be specified
          and the import will only support this one version.

         The following version, if specified with a '[' or '(' indicates the
lower bound.  This can
          be a full semantic version or a MAJOR only or MAJOR.MINOR only.

         The '-', if specified, is a literal hyphen indicating a range will be
specified.  If the second portion
          of the import-versions clause is omitted, then there is no upper
bound on what will be considered
          an acceptable imported version.

         After the '-' the upper bound semantic version (or part thereof)
follows.
         After the upper bound version, one of ']' or ')' MUST follow to
indicate whether this limit is inclusive
          or exclusive of the upper bound respectively.

         Finally, a literal comma (',') MAY be specified with additional
ranges.  Each range is taken as a logical
          OR.

         The statement MUST only be a substatement of the import statement.
Zero or one
```

```
         import-versions statement is allowed per import statement.  NO
substatements are allowed.";
      reference "I-D.clacla-netmod-yang-model-update : Import By Semantic
Version";
    }

    extension status-description {
      argument description;
      description
        "Freeform text that describes why a given node has been deprecated or
made obsolete.
         This may point to other schema elements that can be used in lieu of
the given node.
```

```
         This statement MUST only be used as a substatement of the status
statement, and MUST
         only be used when the status is deprecated or obsolete.  Zero or more
status-description
         statements are allowed per parent statement.  NO substatements are
allowed.";
      reference "I-D.clacla-netmod-yang-model-update : Deprecated and Obsolete
Reasons";
    }

    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
      description
        "Augmentations for the ietf-yang-library module to support semantic
versioning.";
      leaf module-version {
        type string {
          pattern '[0-9]+\.[0-9]+\.[0-9]+';
        }
        description
          "The semantic version for this module in MAJOR.MINOR.PATCH format.
This version
           must match the semver:module-version value in specific revision of
the module
           loaded in this module-set.";
      }
      leaf deprecated-nodes-present {
        type boolean;
        default "true";
        description
          "A boolean that indicates whether or not this server implements
deprecated nodes.
           The value of this leaf SHOULD be true; and if so, the server MUST
implement nodes
           within this module as they are documented.  If specific deprecated
nodes are not
           implemented as document, then they MUST be listed as deviations.  If
a module does
           not currently contain any deprecated nodes, then this leaf SHOULD be
set to true.";
      }
      leaf obsolete-nodes-present {
        type boolean;
        default "false";
        description
          "A boolean that indicates whether or not this server implements
obsolete nodes.
           The value of this leaf SHOULD be false; and if so, the server MUST
NOT implement
```

```
           nodes within this module. If this leaf is true, then all nodes in
this module MUST
           be implemented as documented in the module.  Any variation of this
MUST be listed as
           deviations.  If a module does not currently contain any obsolete
nodes, then this
           leaf SHOULD be set to true.";
      }
    }
    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module/
yanglib:submodule" {
      description
        "Augmentations for the ietf-yang-library module/submodule to support
semantic versioning.";
      leaf submodule-version {
        type string {
          pattern '[0-9]+\.[0-9]+\.[0-9]+';
        }
        description
```

```
          "The semantic version for this submodule in MAJOR.MINOR.PATCH
format.  This version
           must match the semver:module-version value in specific revision of
the submodule
           loaded in this module-set.";
      }
    }
  }
<CODE ENDS>
```

5. Contributors

- Anees Shaikh, Google
- Rob Shakir, Google

6. Security Considerations

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

7. IANA Considerations

7.1. YANG Module Registrations

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

The ietf-semver module:

- Name: ietf-semver
- XML Namespace: urn:ietf:params:xml:ns:yang:ietf-semver
- Prefix: semver
- Reference: [RFCXXXX]

8. References

8.1. Normative References

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

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

8.2. Informative References

[I-D.clacla-netmod-model-catalog]
Clarke, J. and B. Claise, "YANG module for yangcatalog.org", draft-clacla-netmod-model-catalog-03 (work in progress), April 2018.

[I-D.claise-semver]
Claise, B., Barnes, R., and J. Clarke, "Semantic Versioning and Structure for IETF Specifications", draft-claise-semver-02 (work in progress), January 2018.

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.

[I-D.openconfig-netmod-model-catalog]
Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

[openconfigsemver]
"Semantic Versioning for Openconfig Models", <http://www.openconfig.net/docs/semver/>.

[RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <https://www.rfc-editor.org/info/rfc8049>.

[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>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <https://www.rfc-editor.org/info/rfc8299>.

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

[yangcatalog]
"YANG Catalog", <https://yangcatalog.org>.

Appendix A. Appendix

A.1. Open Issues

There are a number of open issues to be disccused. These include the following:

- Do we need a new version of YANG?
  While eventually this will fold into a new version, the belief is this solution can work with extensions alone with an update to the [RFC7950] text concerning module updates.

- Should IETF/IANA officially generate derived semantic versions for their own modules? As they are the owner of the modules it should be their responsibility, but how to document it? Note that next round of funding for the yangcatalog.org could help develop the perfect derived-semantic-version toolset

- We could consider a new naming convention for module files. Today, module files are named using a module@revision.yang notation. We could consider module%semver.yang or module#version.yang variants. Re-using the '@' for version is not ideal, so another separator character should be used. In this manner, both version and revision could be used.

- Taking another page from Openconfig, the notion of a module bundle could be considered. That is, there may need to be a way to enumerate modules that are part of a bundle and are known to interoperate. This may not be as critical if a rich import-by-version is defined.
  While the issue is interesting, it will be not be handled in this document.

- Similarly, the concept of a feature bundle should be considered. Typically, operators combine and test YANG modules to build value-add services. These bundles form releases for specific features or services, and it is critical to ensure as the modules evolve, the bundles can coherently evolve with them.
  While the issue is interesting, it will be not be handled in this document.

- When we'll start using this new procedure for a new YANG module revision, will we have to update all the dependent YANG modules to start using this new procedure, along with the new import statement? Is this a moot point, as a new YANG module name would suffer from the same symptoms?
  We see no need for updating other dependent modules. It is a good idea to update them, as they will benefit from using SEMVER, however there is no specific need to update them.

Authors' Addresses

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Joe Clarke
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

Balazs Lengyel
Ericsson
Magyar Tudosok Korutja
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
United States of America

Email: kd6913@att.com