YANG Module Versioning Requirements

Abstract

This document describes the problems that can arise because of the
YANG language module update rules, that require all updates to YANG
module preserve strict backwards compatibility. It also defines the
requirements on any solution designed to solve the stated problems.
This document does not consider possible solutions, nor endorse any
particular solution.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This requirements document initially considers some of the existing YANG module update rules, then describes the problems that arise due to those rules embracing strict backwards compatibility, and finally defines requirements on any solution that may be designed to solve these problems by providing an alternative YANG versioning strategy.

2. Background

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."

The rules described above, along with other similar rules, causes
various problems, as described in the following sections:

2.1. Striving for model perfection

The points made above 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 Backwards-Compatible

As we learn from our mistakes, we're going to face more and more
non-backwards-compatible YANG modules. An example is the YANG data
model for L3VPN service delivery [RFC8049] , which, based on
implementation experience, has been updated in a non-backwards-
compatible 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), which is sometimes the case for vendor modules
[RFC8199] . From time to time, the new YANG modules are not
backwards-compatible.

Old module parts that are no longer needed, no longer supported, or
are not used by consumers need to be removed from modules. It is
often hard to decide which parts are no longer needed/used; still
the need and practice of removing old parts exist. While it is rare
in standard modules it is more common in vendor YANG modules where
the usage of modules is more controlled.

The problems described in Section 2.7 may also result in
incompatible changes.

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

As modules are sometimes updated in an incompatible way the current assumption that once a YANG module is defined all further revisions can be freely used as they are compatible is not valid.

2.3. Non-Backwards-Compatible Errors

Sometimes small errors force us to make non-backwards-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 a non-backwards-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 backwards-compatible change, just like a correction of the description statement. Current YANG rules are ambiguous as to whether non-backwards-compatible bug fixes are allowed without also requiring a module name change.

2.4. No way to easily decide whether a change is Backwards-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 cannot 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 backwards-compatible way or not.

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

  *It is sometimes necessary 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 backwards-compatible just by looking at the YANG
   statements. It's only the human model designer who can decide.

  *Problems with the deprecated and obsolete status statement,
   Section 2.7

  *YANG module authors 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 comparison of the old and new modules is a tedious task.

2.5. No good way to specify which module revision to import

If a module (MOD-A) is imported by another one (MOD-B) the importer may specify which revision must be imported. Even if MOD-A is updated in a backwards-compatible way not all revisions will be suitable, e.g., a new MOD-B might need the newest MOD-A. However, both specifying or omitting the revision date for import leads to problems.

If the import by revision-date is specified

  *If corrections are made to MOD-A these would not have any effect as the import's revision date would still point to the un-corrected earlier YANG module revision.

  *If MOD-A is updated in a backwards-compatible way because another importer (MOD-C) needs some functionality, the new MOD-A could be used by MOD-B, but specifying the exact import revision-date prevents this. This will force the implementers to import two different revisions of MOD-A, forcing them to maintain old MOD-A revisions unnecessarily.

  *If multiple modules import different revisions of MOD-A the human user will need to understand the subtle differences between the different revisions. Small differences would easily lead to operator mistakes as the operator will rarely check the documentation.

  *Tooling/SW is often not prepared to handle multiple revisions of the same YANG module.

If the import revision-date is not specified

  *any revision of MOD-A may be used including unsuitable ones. Older revisions may be lacking functionality MOD-B needs. Newer MOD-A revisions may obsolete definitions used by MOD-B in which case these must not be used by MOD-B anymore.

  *As it is not specified which revisions of MOD-A are suitable for MOD-B. The problem has to be solved on a case by case basis studying all the details of MOD-A and MOD-B which is considerable work.

2.6. Early Warning about Removal

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 must still be present and usable in the current
revision; however, it will be removed in one of the next revisions.
The deprecated statement cannot be reliably used for this purpose
both because deprecated nodes may not be implemented and also there
is no mandate that text be provided explaining the deprecation.

We need the advance warning to allow users of the module time to
plan/execute migration away from the deprecated functionality.
Deprecation should be accompanied by information whether the
functionality will just disappear or that there is an alternative,
possibly more advanced solution that should be used.

Vendors use such warnings often, but the NMDA related redesign of
IETF modules is also an example where it would be useful for IETF.
As another example, see the usage of deprecated in the Java
programming language.

2.7. 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?

3. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119] .

In addition, this document uses the following terminology:

  *YANG module revision: An instance of a YANG module, with no
   implied ordering or backwards compatibility between different
   revisions of the same module."

  *YANG module version: A YANG module revision, but also with an
   implied partial ordering relationship between other versions of
   the same module. Each module version must be uniquely
   identifiable.

  *Non-backwards-compatible (NBC): In the context of this document,
   the term 'non-backwards-compatible' refers to a change or set of
   changes between two YANG module revisions that do not adhere to
   the list of allowable changes specified in Section 11 "Updating a
   Module" of [RFC7950] , with the following additional
   clarification:

     -Any addition of, or change to, a "status" statement that
      allows a server to remove support for a schema node is
      considered a non-backwards-compatible change

4. The Problem Statement

Considering the issues described in the background, the problem
definition can be summarized as follows.

Development of data models for a large collection of communication
protocols and system components is difficult and typically only
manageable with an iterative development process. Agile development
approaches advocate evolutionary development, early delivery, and
continual improvement. They are designed to support rapid and
flexible response to change. Agile development has been found to be
very successful in a world where the objects being modeled undergo
constant changes.

The current module versioning scheme relies on the fundamental idea
that a definition, once published, never changes its semantics. As a
consequence, if a new definition is needed with different non-
backwards-compatible semantics, then a new definition must be
created to replace the old definition. The advantage of this
versioning scheme is that a definition identified by a module name
and a path has fixed semantics that never change. (The details are a
bit more nuanced but we simplify things here a bit in order to get
the problems worked out clearly.)

There are two main disadvantages of the current YANG versioning scheme:

  *Any non-backwards-compatible change of a definition requires
   either a new module name or a new path. This has been found
   costly to support in implementations, in particular on the client
   side.

  *Since non-backwards-compatible changes require either a new
   module name or a new path, such changes will impact other modules
   that import definitions. In fact, with the current module
   versioning scheme other modules have to opt-in in order to use
   the new version. This essentially leads to a ripple effect where
   a non-backwards-compatible change of a core module causes updates
   on a potentially large number of dependent modules.

Other problems experienced with the current YANG versioning scheme
are the following:

  *YANG has a mechanism to mark definitions deprecated but it leaves
   it open whether implementations are expected to implement
   deprecated definitions and there is no way (other than trial and
   error) for a client to find out whether deprecated definitions
   are supported by a given implementation.

  *YANG does not have a robust mechanism to document which data
   definitions have changed and to provide guidance how
   implementations should deal with the change. While it is possible
   to have this described in general description statements, having
   these details embedded in general description statements does not
   make this information accessible to tools.

  *YANG data models often do not exist in isolation and they
   interact with other software systems or data models that often do
   allow (controlled) non-backwards-compatible changes. In some
   cases, YANG models are mechanically derived from other data
   models that do allow (controlled) non-backwards-compatible
   changes. In such situations, a robust mapping to YANG requires to
   have version numbers exposed as part of the module name or a path
   definition, which has been found to be expensive on the client
   side (see above).

Given the need to support agile development processes and the
disadvantages and problems of the current YANG versioning scheme
described above, it is necessary to develop requirements and
solutions for a future YANG versioning scheme that better supports
agile development processes, whilst retaining the ability for
servers to handle clients using older versions of YANG modules.

5. Requirements of a YANG Versioning Solution

The following is a list of requirements that a solution to the problems mentioned above MUST or SHOULD have. The list is grouped by similar requirements but is not presented in a set priority order.

1. Requirements related to making non-backwards-compatible updates to modules:

    1.1   A mechanism is REQUIRED to update a module in a non-backwards-compatible way without forcing all modules with import dependencies on the updated module from being updated at the same time (e.g. to change its import to use a new module name).

    1.2   Non-backwards-compatible updates of a module MUST not impact clients that only access data nodes of the module that have either not been updated or have been updated in backwards-compatible ways.

    1.3   A refined form of YANG's 'import' statement MUST be provided that is more restrictive than "import any revision" and less restrictive than "import a specific revision". Once non-backwards-compatible changes to modules are allowed, the refined import statement is used to express the correct dependency between modules.

    1.4   The solution MUST be able to express when non-backwards-compatible changes have occurred between two revisions of a given YANG module.

2. Requirements related to identifying changes between different module revisions:

    2.1   Readers of modules, and tools that use modules, MUST be able to determine whether changes between two revisions of a module constitute a backwards-compatible or non-backwards-compatible version change. In addition, it MAY be helpful to identify whether changes represent bug fixes, new functionality, or both.

    2.2   A mechanism SHOULD be defined to determine whether data nodes between two arbitrary YANG module revisions have (i) not changed, (ii) changed in a backwards-compatible way, (iii) changed in a non-backwards-compatible way.

3. Requirements related to supporting existing clients in a backwards-compatible way:

    3.1

The solution MUST provide a mechanism to allow servers to support existing clients in a backwards-compatible way.

   3.2   The solution MUST provide a mechanism to support clients that expect an older version of a given module when the current version has had non-backwards-compatible changes.

   3.3   Clients are expected to be able to handle unexpected instance data resulting from backwards-compatible changes.

4. Requirements related to managing and documenting the life cycle of data nodes:

   4.1   A mechanism is REQUIRED to allow a client to determine whether deprecated nodes are implemented by the server.

   4.2   If a data node is deprecated or obsolete then it MUST be possible to document in the YANG module what alternatives exist, the reason for the status change, or any other status related information.

   4.3   A mechanism is REQUIRED to indicate that certain definitions in a YANG module will become status obsolete in future revisions but definitions marked as such MUST still be implemented by compliant servers.

5. Requirements related to documentation and education:

   5.1   The solution MUST provide guidance to model authors and clients on how to use the new YANG versioning scheme.

   5.2   The solution is REQUIRED to describe how to transition from the existing YANG 1.0/1.1 versioning scheme to the new scheme.

   5.3   The solution MUST describe how the versioning scheme affects the interpretation of instance data and references to instance data, for which the schema definition has been updated in a non-backwards-compatible way.

6. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following people are members of that design team and have contributed to defining the problem and specifying the requirements:

  *Balazs Lengyel

*Benoit Claise

*Ebben Aries

*Jason Sterne

*Joe Clarke

*Juergen Schoenwaelder

*Mahesh Jethanandani

*Michael (Wangzitao)

*Qin Wu

*Reshad Rahman

*Rob Wilton

7. Acknowledgments

The design team would like to thank Christian Hopps and Vladimir
Vassilev for their feedback and perspectives in shaping and fine
tuning the versioning requirements.

One of the inspirations for solving the YANG module versioning comes
from OpenConfig. The authors would like to thank Anees Shaikh and
Rob Shakir for their helpful input.

8. Security Considerations

The document does not define any new protocol or data model. There
is no security impact.

9. IANA Considerations

None

10. References

10.1. Normative References

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

[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>.

10.2. Informative References

[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>.

Author's Address

Joe Clarke (editor)
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com