Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 January 2024

J. Schönwälder
Constructor University

24 July 2023

YANG 1.1 with Relaxed Versioning (YANG 1.2) draft-schoenw-netmod-yang-relaxed-versioning-00

Abstract

This document relaxes the module update rules of the YANG data modeling language allowing authors to make non-backwards compatible changes in limited situations. The YANG 1.1 specification together with the extensions and updates defined in this document defines YANG version 1.2.

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 25 January 2024.

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC7950] is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF) [RFC6241]. Section 11 of [RFC7950] defines the rules for updating a YANG module. The update rules were designed to avoid interoperability problems:

> [...] 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.

Section 11 details a number of allowed changes and then concludes with the following general rule:

> 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.

In the real world, there are situations where the cost of introducing a new definition do not align with the risk of making a non-backwards compatible change. The relaxed versioning rules allow limited non-backwards compatible changes if the costs of introducing new definitions significantly outweigh the costs associated with the risk to breaking deployed systems.

The following changes to the YANG 1.1 language are made:

1. This document defines an import by minimum revision mechanism, which improves the existing overly restrictive import by exact revision statement.

2. This document introduces annotations to indicate in which version of a YANG module definitions were added, updated, removed, or semantically changed.

3. This document relaxes the YANG module update rules to allow non-backwards compatible changes in situations where the full implications on deployed clients are understood and the risks have been carefully weighed against the costs of introducing new definitions.

An implementation of [RFC7950] also implementing this specification is called YANG 1.2.

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

2. Import and Include by Minimum Revision

The YANG 1.1 import and include statements are extended by allowing a "min-revision-date" substatement defining the minimum revision that an imported or included module must have.

```
import foo-module {
  min-revision-date 2023-04-01;
}
```

DISCUSS: It may make sense to also introduce 'max-revision-date' as this may be necessary eventually.

3. Annotations to Indicate Changes

The YANG language is extended with statements that indicate when definitions were added, when definitions where updated, when the status of definitions changed, and when definitions changed in a non-backwards compatible way. Furthermore, the status statement is revised to allow for substatements, namely description statements and statements detailing when additions or changes were made.

Here is an example. Lets assume that a leaf 'foo' is added to a module after its first publication:

```
    leaf foo {
      type int16;
      description
       "A number identifying a foo.";
      since 2021-12-06;
    }
```

Note that this is a shorthand notation expanding to the following:

```
    leaf foo {
      type int16;
      description
       "A number identifying a foo.";
      status current {
        since 2021-12-06;
      }
    }
```

The 'since' statement defaults to the initial version of a module if it is not present, hence no 'since' statements are needed for definitions that were part of the initial published version of a module.

Lets assume that using a signed integer type was later found to be a mistake. The robust and default path forward is the introduction of a new leaf 'bar' and the deprecation of the old leaf 'foo'.

```
    leaf bar {
      type uint16;
      description
       "An unsigned number identifying a foo.";
      since 2023-04-01;
    }

    leaf foo {
      type int16;
      description
       "A number identifying a foo.";
      since 2021-12-06;
      status deprecated {
        since 2023-04-01;
        description
         "This leaf uses signed values, which has led to
          interoperability problems. Implementations should
          use the new bar leaf instead, which uses an unsigned
          number space.";
       }
     }
```

Modules can use the import by minimum revision mechanism to import version 2023-04-01 (or later) and then make use of 'bar' instead of 'foo'.

In the special and rare situation where replacing 'foo' with 'bar' is considered too costly relative to the risks of breaking deployed systems, the following non-backwards compatible change can be made

```
leaf foo {
  type uint16;
  description
   "An unsigned number identifying a foo.";
  status current {
    since 2021-12-06;
    changed 2023-04-01 {
       description
        "The leaf used signed int16 values, which has led
         to interoperability problems. This got fixed by
         changing to an unsigned type. This potentially
         breaks systems that use negative values. Such
         systems are considered to be rare if they exist
         at all.";
    }
  }
}
```

DISCUSS: More clearly distinguish between backward compatible updates and non-backwards compatible changes. Right now, I am thinking of using 'updated' for backwards compatible changes and 'changed' for non-backwards compatible changes but perhaps this is too subtle?

4. Relaxing Module Update Rules

The following statement in Section 11 of RFC 7050

> 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.

is replaced by the following text:

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 SHOULD be achieved by a new definition with a new identifier. In exceptional and rare cases, where the costs of introducing a new definition clearly outweighs the risks of potentially breaking deployed implementations, a non-

backwards compatible change can be made. The appropriate annotation statements MUST be used in this situation to clearly describe the non-backwards compatible change.

5. YANG ABNF Grammar Extensions

TODO

6. Security Considerations

This document defines an extension of the YANG data modeling language. The definitions themselves have no security impact on the Internet, but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [RFC7950] apply for this document as well.

7. Acknowledgments

Helpful comments on various versions of this document were provided by the following individuals: TBD

8. 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>.

[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>.

9. Informative References

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

Author's Address

Jürgen Schönwälder
Constructor University
Email: jschoenwaelder@constructor.university