

Module Versioning: Imports

IETF NETMOD Interim
Dec 14, 2020

Introduction

Two issues related to versioning and imports:

1) Impact of NBC changes on imports. Do we need import by revision-or-derived-compatible?

➤ <https://github.com/netmod-wg/yang-ver-dt/issues/75>

2) Impact of changing an import statement -> BC or NBC?

➤ <https://github.com/netmod-wg/yang-ver-dt/issues/4>

Impact of NBC changes on imports

- Import sub-statement extension “*revision-or-derived*” was introduced in module-versioning to alleviate the restrictions of import by date. It reduces the set of importable revisions to those which are derived from a particular revision
- Consider module A (1.0.0) which imports module B using “2.0.0 or derived” and that there is no revision-label with MAJOR version > 2. This means A will be importing rev 2.Y.Z of module B.
- If new revision 3.0.0 of module B is created (NBC changes), module A may end up importing 3.0.0 and this **could** break clients using module A. It’s also possible module A does not want the changes made in 3.0.0 of module B.
- Should we also have another extension “*revision-or-derived-compatible*” to limit the import set to BC revisions? e.g “2.0.0 or derived compatible” would limit the imported version to 2.Y.Z, 3.0.0 would NOT be a candidate. Note that this would be done by looking at the revision history: revisions after 2.0.0 which are marked NBC via the *rev:nbc-changes* extension would be excluded.

Reminder on import

RFC7950:

```
import module-b {  
    revision-date 2018-04-02; // specific revision  
}
```

draft-ietf-netmod-module-versioning:

```
import module-b {  
    revision-or-derived 2.0.0 ; // revision 2.0.0 or any descendent  
}
```

What we are considering:

```
import module-b {  
    revision-or-derived-compatible 2.0.0; // revision 2.0.0 or any descendent compatible with 2.0.0  
}
```

Example 1: obsoleting if-index from ietf-interfaces

- Consider scenario where if-index is deprecated and eventually obsoleted. Adding the obsolete status is an NBC change, and ietf-interfaces would e.g. go from version 2.x.y to 3.0.0
- With “*revision-or-derived 2.0.0*”, all importing modules would be able to import the new version automatically
- With “*revision-or-derived-compatible 2.0.0*”, all importing modules would be stuck importing 2.x.y. They would need to be modified to be able to import 3.0.0

Example 2: changing a type in an imported module

- Module B 2.0.0 has a grouping containing node *vpn-id* as an *integer*. Module A uses that grouping.
- In 3.0.0 of module B, *vpn-id* is modified to be a *string*
- Some servers/implementations may want to keep *vpn-id* as *integer* while others may desire the new *string* definition
- With “*revision-or-derived 2.0.0*”, all importing modules would get the new definition
- With “*revision-or-derived-compatible 2.0.0*”, all importing modules would keep the old definition
- Both statements are useful. Module A could be branched accordingly.

revision-or-derived-compatible

➤ Pros

- No accidental breakage to an importing module due to an NBC change in an imported module i.e. the owner of an importing module has control
- We know exactly what major version is being used and the impact of changing the major version is clear
- Can be used for reactive repair of including module if newer version of included module breaks the including module (e.g grouping removed) or is not desired.

➤ Cons

- If import of an NBC revision is desired, this requires modification of many importing modules. This is similar to import by date
- Do not automatically get NBC fixes made to imported modules
- Potentially confusing to have 2 flavours of the import by derived substatement. Module owners may pick one not fully understanding the implications

Impact of changing import stmt

- Consider module A (1.0.0) which imports module B using “2.0.0 or derived” and that there are revision-labels with MAJOR version 2 and 3. This means A will import rev 2.Y.Z or 3.Y.Z of module B.
- If module A is modified to importing module B using “3.0.0 or derived”, is this a BC or NBC change?
- **Authors/contributors believe that a change to an import statement should always be considered to be a BC change to the importing module.**
- The revision label of a module represents the schema defined in *that module*.
- Clients know all the module versions in the schema (via YANG packages or YANG Library). The NBC change in module B is reported in the schema, no need to also change the version of module A
- The revision-label of the corresponding YANG package is updated according to the impact on the package’s schema.

Impact of changing import stmt (other option considered)

- We also considered changing the version of module A depending on the BC/NBC impact but have the following concerns:
 - Potential ripple effect, e.g. if module A includes module B which includes C etc etc, changing one import statement at the bottom could lead to many modules having their version updated to reflect NBC change.
 - There is no need to reflect the NBC change on including modules since clients have to look at the whole schema