NETMOD Module Versioning Update

NETMOD WG

Nov 2022

Presenting on behalf of the weekly versioning call attendees:
Jason Sterne

IETF 115

Recent Updates

https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning

Released -07 since IETF 114. Changes include:

- reworked the introduction as per WG LC feedback (incl. a summary of the 5 drafts) [Issues #145 and #146]
- renamed the module level NBC extension to non-backwards-compatible-revision to more clearly differentiate vs the per-node extensions in the Schema Comparison draft
- removed the bullet in section 3.1.1 allowing reordering of YANG statements as a BC change. We'll stick with RFC7950 on this one (reordering is NBC) [Issue #153]
- Clarified that changes to import statements are classified as BC (author always has the option to bump that to NBC if they want to) [Issue #4]
- Filename format acme-router-module#2.0.3.yang RECOMMENDED if label available (updates RFC6020/7950/8407) [Issue #148]
- Clarified what it means to reduce/change the value set of a leaf [Issue #175]

Per Node Compatibility Indicators

- 3 indicators – all optional:
  - nbc-change-at
  - bc-change-at
  - editorial-change-at
- Recommended when a schema comparison tool may not identify a change correctly (e.g. a pattern change that is actually BC, a description change that is actually NBC)

```
leaf used-to-be-a-string {
  rev:nbc-change-at "3.0.0" {
    description "Changed from a string to
    a uint32.";
  }
  type uint32;
}
…
list sir-changed-a-lot {
  rev:editorial-change-at "3.0.0";
  rev:bc-change-at "2.3.0";
  rev:bc-change-at "1.2.1_non_compatible";
  description "a list of stuff";
  ordered-by user;
  key "foo";
  …
}
```

- Not part of base Module Versioning draft (TBD: Part of Schema Comparison draft ? split into separate draft ?)
  - The module level NBC marker is needed in any case, and has broad interest (O-RAN, OpenConfig)
  - Per node markers are more complex. More of an additional enhancement on top of the basics.
  - Per node markers fit better with the schema comparison / tooling topic
  - Keep the body of module as the API rather than clutter it with embedded history/versioning information

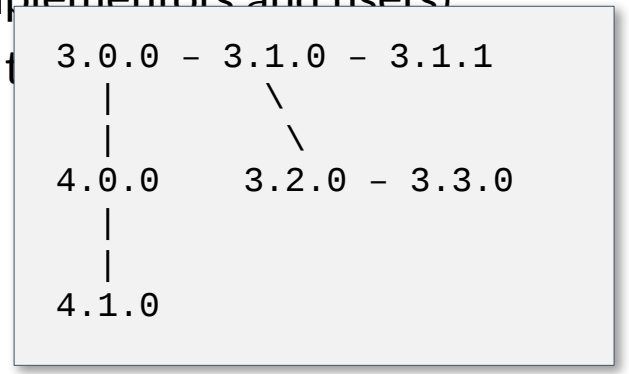
Outstanding Issues

- Under discussion: making import by revision-or-derived less strict (hint instead of rule)
- Half dozen updates required from WG LC (captured in GitHub Issues)
- WG review & discussion of per-node compatibility extensions (as part of Schema Comparison)

Softening by revision-or-derived

- Considering softening as "suggested minimum revision" instead of current hard "minimum required revision"
  - i.e. allow the use of a revision that isn't derived if a conforming revision isn't available
- When YANG library advertises multiple revisions of a module: our draft says to select the "implemented" version (not the latest), or the latest "import only" version if none are implemented.
  - how should this interact with import by revision-or-derived ?
- Conformance vs Compiler:
  - 7950 today is more about YANG library and conformance
  - revision-or-derived text was somewhat focused on compiler behavior

- Does it even matter if this is a hard vs soft rule – library advertises a set of modules and does it matter if this constraint is met?
- Recommendation/Documentation instead of rule (but still very useful for module implementors and users)
- Tools allowed to import other revisions - but SHOULD (MUST?) use the suggestion if t
  - flexibility vs some ambiguity (note: same ambiguity exists today)
- May work better with branched module development



BACKUP SLIDES FROM IETF114

Why do import by derived revision this way?

Most common issue: I need to import any version that includes item X added in version Z; I import/use the item X

- Earlier version are not usable, later versions are probably all usable
- I want to follow the updates of the imported module without updating the importers
- Tradeoff between precise but complicated & restrictive solution and simple & but risky imports

Pitfalls to avoid:

- If the imported module is updated in an NBC way item X may be removed/changed:
  - rare case we accept this risk, mostly detected in compile time
- Only accept compatible imports: if a module defines 20 items and one is updated NBC the importers of the other 19 items also need to be updated
- Definition of allowed versions can become extremely complicated
  - E.g. Ranges, inclusive and exclusive filters
    see draft-clacla-netmod-yang-model-update#section-2.2

```
//earlier abandoned complex solution
import example-module {
  semver:import-versions "[1.1-2),[3";
}
```

Examples where import by derived revision is needed:

- ietf-yang-types – need to ensure that newly added types are present
- _3gpp-common-yang-types – as above

- iana modules like algorithm types, if-types, hw-types – as above
- ietf-yang-library – only new version list datastores e.g. required in RFC9196

Why do we need import by derived revision?

Current situation:

Plain import – any revision can be used.

- Bad because e.g., an old (RFC6021) revision of ietf-yang-types could be used not containing newer types like hex-string, which the importer needs from RFC6991; or missing critical corrections

Import by exact revision

- Bad because
  - the importer needs updates if the imported module is corrected for faults.
  - A system where many modules import one module might need to support many versions of the imported module
  - Importers will need frequent updates, updates which are hard to track.
  - Practically not used in IETF or elsewhere

Import by derived revision - new

- Better because
  - Can ensure new functionality is included
  - Doesn't need updates to the importer - usually
  - Importer and imported versions are only loosely coupled
  - Derived means not just later, but derived according to the development path (revision statements)
  - If the compiler doesn't know this extension, it is no worse than a plain import. As an example
    - Old compiler can choose any of 5 versions
    - New compiler knows that the first 2 versions are unsuitable

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