NETMOD Module Versioning Update

NETMOD WG

Jul 2022

Presenting on behalf of the weekly versioning call attendees:
Balazs Lengyel

IETF 114

Current Status – Outstanding issues

- Released -06 with minimal updates

- Individual data node change annotations
  - Allow author to declare formally NBC changes as BC
- Import by derived revision
- File Naming
- YANG 1.1 or YANG-Next

Individual data node change annotations Background

- Module versioning draft specifies “rev:non-backwards-compatible” marker in **module** revision history:
  - Indicates **some** NBC changes have occurred **somewhere** in the module between revisions.
  - If also using YANG Semver, then version number change will **also** indicate the NBC change
- Some WG LC feedback that a per data-node* change annotation to indicate nbc-changes would be more helpful for readers.
- Author’s consensus: Both are helpful, but serve different purposes

Module level revision annotations

Note - already presented in the overview …

A) Mark a revision as non-backwards-compatible

```
module example-module {

    namespace "urn:example:module";
    prefix "prefix-name";

    import ietf-yang-revisions { prefix "rev"; }
    import ietf-yang-semver { prefix "ysver"; }

    rev:revision-label-scheme "ysver:yang-semver";

    revision 2019-06-01 {
      rev:revision-label 3.1.0;
      description "Add new functionality.";
    }

    revision 2019-03-01 {
      rev:revision-label 3.0.0;
      rev:non-backwards-compatible;
      description
        "Add new functionality. Remove some deprecated nodes.";
    }

    revision 2019-02-01 {
      rev:revision-label 2.0.0;
      rev:non-backwards-compatible;
      description "Apply bugfix to pattern statement";
    }
```

Example of a per data-node change annotation

- This example is of a single simple nbc-change annotation.
- Equivalent bc-change and editorial-change annotations can also be defined.
- The date binds the change to the specific module revision where it occurred
  - allowing a label instead of a date is probably helpful.

```
leaf foo {
  type string;
  nbc-change "2022-03-01" {
    description "changed type from int32 to string";
  }
  // ...
}
```

Alternative proposal received during WG LC:

- Authors MUST add per data-node NBC change markers for all NBC changes:
  - No annotation => not an NBC change (i.e., BC or editorial)
  - Module consumers must/can scan module to determine change history

- Author’s concerns:
  - Module could end up cluttered with full per data-node history
  - Hard for tooling to spot mistakes (e.g., pattern change looks like a nbc-change but hasn’t been flagged). Is this because the author forgot, or because it is a bc-change?
  - Can we convince vendors and OpenConfig, 3GPP to follow this path? If not then clients left in completely ambiguous situation.
    - Other SDOs working with many releases per year will have many NBC changes
  - How to handle deleted nodes?
  - Still need module level annotation regardless

Weekly call group consensus:

- Per data-node change annotations are REQUIRED only whenever a comparison tool cannot reasonably determine the right answer
- Module authors MAY* add change annotations for clarity
- No change annotation implies nothing

Aim is to:

- minimize unnecessary annotations (module files reflect snapshot of current API, not full evolution)
- use tooling (e.g., a pyang plugin) to accurately* compare versions and classify and report changes

When are per data-node change annotations used?

- Mostly, tooling can apply the rules in yang-module-versioning to determine type of change. In some cases:
  - tooling cannot reliably do this, or may default to the wrong choice
  - so the module author MUST add per data-node bc/nbc change marker to override tool's default
- Changes in different YANG statements have different tool defaults:
  - Based on practicality of evaluating complex yang-module-versioning rules
  - E.g., must, when, pattern => assume nbc if tool detects these have changed
  - description, reference => assume bc (or editorial) if tool detects these have changed
- Module authors MAY always add change annotations for clarity

Where to define per node annotations ?

- Where should per data-node change annotation extensions be specified?
  - Currently planned for draft-ietf-netmod-yang-schema-comparison
  - Alternatively could be brought into yang-module-versioning
  - However, more refinement expected, so would likely delay module versioning a bit.

- Weekly call preference?
  - Not entirely set, leaning towards bringing the annotations in? But don't want significant delay to drafts.

Declare formally NBC change as BC ?

Do we need to allow annotations to flag formally NBC changes as BC? – YES, sometimes that's the reality

https://github.com/netmod-wg/yang-ver-dt/issues/154

Examples where it is needed:

- pattern statement changed: **restructuring a regexp**, but server **value space didn't change**
- comparison tool may simply treat **complex statements as NBC ;** not practical to determine BC/NBC (pattern, when, must, type change)
  - type int32 { range 0..255; } changed to type uint8;
- change is made in a YANG model to more accurately reflect the actual server constraints that were always there (**Correct mistakes**)
  - server only ever accepted values 1..15 (commit would fail with value 16)
  - old YANG model incorrectly showed a range of 1..16
  - correction done to change the range to 1..15
- A vendor module might live in a "**walled garden**" where all clients are known. In this case a change that is really NBC might not be a problem for the clients: the change is BC for all users.

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;
}
```

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

Import by derived revision and tool behavior

- What should a “pure” RFC7950 tool (compiler/parser) do when it encounters the *revision-or-derived* extension in import statements?
    - Since it does not support the YANG versioning extensions, it will treat these import statements as an RFC7950 import. E.g. the tool may use an incorrect revision even though a correct revision is available.
    - Should the tool instead fail when it encounters the *revision-or-derived* extension since it does not support that extension? We propose that it is preferable to ignore the extension and behave as per RFC7950 section 5.1.1. And if an incorrect revision is imported, the tool may fail anyway, e.g. due to a missing type.
        - Would need the tool to at least recognize that this is a critical extension, already assumes tool updates

- Should we make the *revision-label-scheme* extension mandatory if a module has a revision-label? This would allow tools to fail when they encounter a revision-label scheme which they do not support
    - Proposal make it mandatory for IETF models, and recommended for other models

Revision Label in file name

- For modules which use revision-label, it is very convenient to refer to them by revision-label, as opposed to only by date
- We have the option of having revision-label, qualified with '#', in the filename of a YANG module, e.g. *ietf-xxx#2.1.3.yang* which would have same contents as the corresponding filename with the date e.g. *ietf-xxx@2022-07-18.yang*
- The text currently uses (contradictory?) RFC2119 language
- Proposal for new text:

```
This section updates [RFC7950] section 5.2 and [RFC6020] section 5.2.

If a revision has an associated revision label, then the revision-label can be used instead
of the revision date in the filename of a YANG file, where it takes the form:
  module-or-submodule-name [['@' revision-date]|['#' revision-label]] ( '.yang' / '.yin' )
    E.g., acme-router-module@2018-01-25.yang
    E.g., acme-router-module#2.0.3.yang
YANG module (or submodule) files can be identified using either revision-date or revision-
label. Two file names, one with the revision date and another with the revision label, can
exist for the same module (or submodule) revision, e.g., when migrating from one scheme to
the other.
```

Module Versioning work in YANG 1.1 vs YANG NEXT

- Some WG LC discussions around making some aspects of the YANG Versioning drafts mandatory parts of YANG NEXT

- That may be a good idea

- But that doesn’t preclude making some of it optional in YANG 1.1.
  - At least have a chance of making many things better
  - Even if tools don’t use/understand new extensions, etc, they are no worse off than today

- Consensus of weekly call group: continue with this work as part of YANG 1.1