NETMOD YANG Packages Update

NETMOD WG
March 2022

Presenting on behalf of the weekly versioning call attendees:
Jan Lindblad

IETF 113
General update on YANG Packages
Major Discussions / Topics

Issue #57 - open:
• We do want to include information about mount points in Packages
• Still open debate about mandatory vs optional schema at the mount points

Issues #67/69 - closed:
• Examined in detail how Packages and Library fit together (along with other advertisements of modules & conformance)
• Analyzed an alternative approach to try and reduce overlap but in the end we couldn’t find a significant enough net improvement

Issue #133/#135 - open:
• Allowing modules in Packages to be optional
• API Packages vs Implementation Packages
General update YANG Packages
Minor Discussions/Topics

Issue #65 - closed:
• added revision label scheme for Packages
• IETF MUST use YANG Semver for Packages. All other orgs SHOULD use a revision-label

Issue #70 - closed:
• update text for deviations in Packages

Issue #74 - open:
• fixed ‘schema’ terminology
• fixed pre-release package versions
• some other minor aspects still open until text is rolled into the latest Packages draft
General update YANG Packages
Minor Discussions/Topics cont.

Issue #82 - closed:
• removed checksums from Packages

Issue #105/#125 - closed:
• remove nbc-changes, parent and history
• fix ‘version’ vs ‘revision’ terminology

Issue #138 – closed:
• Allow submodules in packages to be identified by revision-label
Two similar but different uses for YANG packages:

1. To define a management API:
   E.g., IETF could define a routing package, EVPN package, etc. Each package defines the required modules and enabled features. Available offline, encourages conformance and consistency.

2. To define the server implementation:
   List which APIs are supported, deviations, and any extra modules or features that are also implemented. May be available offline, aims to simplify conformance.
Types of YANG Package

Current proposal:

```yang
grouping yang-pkg-instance:
  +-- name                  pkg-name
  +-- version               pkg-version
  +-- pkg-type?             pkg-type
  +-- timestamp?            yang:date-and-time
  +-- ...
  +-- supported-feature*   scoped-feature
  +-- included-package*    [name]
    |  +-- name             pkg-name
    |  +-- version          pkg-version
    |  +-- replaces-version* pkg-version
    |  +-- location*        inet:uri
  +-- module*    [name]
    |  +-- name             yang:yang-identifier
    |  +-- revision?
      |       rev:revision-date-or-label
    |  ...
  +-- import-only-module* [name revision]
    |  +-- name?            yang:yang-identifier
    |  +-- revision?
      |       rev:revision-date-or-label
    |  ...
  +-- implements-package* [name]
    +-- name            pkg-name
    +-- version         pkg-version
    +-- nbc-modified?   boolean
    +-- location*       inet:uri
```

Package type: “api” or “implementation

- Rename “mandatory-feature” to “supported-feature”
- [Only present for implementation packages]
  Lists all implemented API packages
Types of YANG Package

Module: ietf-yl-packages

```yang
augment /yanglib:yang-library/yanglib:schema:
  +--ro package* [name]
    +--ro name       ->
      /pkgs:packages/implementation/package/name
    +--ro version    leafref
```

Module: ietf-yang-packages

```yang
+--ro packages
  +--ro api
    |  +--ro package* [name version]
    |      +--ro name                  pkg-name
    |      +--ro version               pkg-version
    |      +--ro pkg-type?             pkg-type
    |      ...
    +--ro implementation
      +--ro package* [name]
      +--ro name                  pkg-name
      +--ro version               pkg-version
      +--ro pkg-type?             pkg-type
      +-- ... 
      +--ro implements-package* [name]
```

Datastore schema bound to 1+ implementation packages

- Separate list of API packages vs implementation packages
- Client shouldn’t need to fetch API packages because they are available offline
- Implementation packages may also be available offline.
Example of API vs Implementation Package

Example API package:

name: ietf-routing
version: 1.3.1
description:
  "IETF routing package"
includes-package:
  ietf-ntwk-device, 1.1.2,
  ietf-bgp, 2.0.0,
  ietf-isis, 2.3.0
  ...
compatible-package:
  ietf-rip, 1.0.0,
  ietf-vrrp, 2.0.0
  ietf-acls, 1.0.0
  ...

Example implementation package:

name: vendor-router
version: 3.0.0
description:
  "Platform XXXX, S/W R4.5 or later"
supported-feature:
  foo:xxx, bar:yyy
implements-package:
  ietf-routing, 1.3.1
  ietf-ntwk-device, 1.1.2
  ietf-bgp, 2.0.0
  ietf-isis, 2.3.0
  ietf-vrrp, 2.0.0
  ietf-acls, 1.0.0
modules:
  vendor-bgp-deviations, 1.0.0
Types of YANG Package

Questions for the WG:
• Is this split between API vs implementation packages useful?
• Other comments?
Optional modules (issue #133)

For API packages:
   Some functionality may be optional

1. Could mark some included modules/packages as being “optional”
   But, increases complexity, particularly for what it means to “implement” a package.

2. Don’t allow optional modules in a package
   But, could allow extra metadata in a package definition to list “compatible” module and package versions

Still actively being discussed in weekly meetings, currently leaning towards the second choice
Optional Modules (Option 1): Compatible packages/modules

Current proposal:

```
grouping yang-pkg-instance:
  -- name                  pkg-name
  -- version               pkg-version
  -- pkg-type?             pkg-type
  -- timestamp?            yang:date-and-time
  -- ...
  -- supported-feature*    scoped-feature
  -- package* [name]
    -- name                pkg-name
    -- version             pkg-version
    -- replaces-version*   pkg-version
    -- location*           inet:uri
    -- optional            boolean
  -- module* [name]
    -- name                yang:yang-identifier
    -- revision?           rev:revision-date-or-label
    -- optional            boolean
  -- ...
  -- import-only-module* [name revision]
    -- name?              yang:yang-identifier
    -- revision?          rev:revision-date-or-label
    -- ...
  -- implements-package* [name]
    -- name               pkg-name
    -- version            pkg-version
    -- nbc-modified?      boolean
    -- location*          inet:uri
```

- API packages can specify included packages/modules as being “optional”.
- Implementation packages must specify all implemented optional modules/included packages:
  - Implementing an optional package implements all non optional modules/packages in that package.
  - Seems to get complex …
Optional Modules (Option 2): Compatible packages/modules

Current proposal:

```
grouping yang-pkg-instance:
  +-- name                  pkg-name
  +-- version               pkg-version
  +-- pkg-type?             pkg-type
  +-- timestamp?            yang:date-and-time
  +-- ...                      
  +-- supported-feature*    scoped-feature
  +-- package* [name]
      |  +-- name                pkg-name
      |  +-- version             pkg-version
      |  +-- replaces-version*   pkg-version
      |  +-- location*           inet:uri
      |  +-- module* [name]
          |      +-- name                yang:yang-identifier
          |      +-- revision?          rev:revision-date-or-label
          |      ...                   
      +-- import-only-module* [name revision]
      |  +-- name?                yang:yang-identifier
      |  +-- revision?            rev:revision-date-or-label
      ...                       
  +-- implements-package* [name]
      +-- name            pkg-name
      +-- version         pkg-version
      +-- nbc-modified?   boolean
      +-- location*       inet:uri
  +-- compatible-package* [name]
      +-- name        pkg-name
      +-- version     pkg-version
      +-- location*   inet:uri
  +-- compatible-module* [name]
      +-- name        yang:yang-identifier
      +-- revision?   rev:revision-date-or-label
      +-- location*   inet:uri
```

- An [API] package definition can optionally list related compatible packages and modules.
- Does not change the API defined by a package.
- Specifies RECOMMENDED versions of compatible packages/modules if they are going to be implemented along side.
- This information would be optional to include – it could be defined elsewhere out of band.
Optional modules - Questions for the WG:

• Should packages genuinely support optional modules (as per option (1))? Or does that introduce too much complexity?

• Is the alternative proposed solution (2), better? Does “compatible modules/packages” add value, or just complexity?

• Other comments?
Schema mount in packages (issue #57)

• This had been discussed in the weekly meetings
• But the questions related to optional packages/modules at a mount point took the conversation to the more general question of optional modules/packages in a YANG package schema
• Next slide represents one form of the structure we were discussing at the time.
  • Doesn’t take into account package types, or optional/compatible modules
Schema mount in packages

Current proposal:

```
grouping yang-pkg-instance
  ++-- name                  pkg-name
  ++-- version               pkg-version
  ++-- included-package* [name version]  
    |  ++-- name                pkg-name
    |  ++-- version             pkg-version
    |  ++-- replaces-version*   pkg-version
    |  ++-- location*           inet:uri
    |  ...
  ++-- module* [name]       
    |  ++-- name
    |  ...
  ++-- import-only-module* [name revision] 
    |  ++-- name
    |  ...
  ++-- schema-mounts
    ++-- mount-point* [module label]  
      |  ++-- module?   yang:yang-identifier
      |  ++-- label?    yang:yang-identifier
      |  ++-- config?   Boolean
      |  ++-- package* [name]    
        |  ++-- name        pkg-name
        |  ++-- version     pkg-version
        |  ++-- location*   inet:uri
```

- An API package could include “design time” information about what mounted schema could/should be available.
- An implementation package could include “design time” or “runtime” information about what schema is (or will be) available at a given mount point.
  - I.e., a vendor can indicate offline that they will find BGP, OSPF and ISIS under the VRF mount point in the network instances model.
- Some open questions are:
  - Is this too much complexity?
  - Are listed mounted packages optional to implement (see *)? Can clients find other mounted modules/packages, in addition to, or instead of, the listed packages?
  - Do we need to tweak “compatible packages” scheme to work with mount points as well?