YANG Tree Diagrams
draft-ietf-netmod-yang-tree-diagrams-02

Abstract

   This document captures the current syntax used in YANG module Tree
   Diagrams. The purpose of the document is to provide a single
   location for this definition. This syntax may be updated from time
   to time based on the evolution of the YANG language.

Table of Contents

1. Introduction

   YANG Tree Diagrams were first published in [RFC7223]. Such diagrams
   are commonly used to provided a simplified graphical representation
   of a data model and can be automatically generated via tools such as
   "pyang". (See <https://github.com/mbj4668/pyang>). This document
   provides the syntax used in YANG Tree Diagrams. It is expected that
   this document will be updated or replaced as changes to the YANG
   language, see [RFC7950], necessitate.

   Today's common practice is include the definition of the syntax used
   to represent a YANG module in every document that provides a tree
   diagram. This practice has several disadvantages and the purpose of
   the document is to provide a single location for this definition. It
   is not the intent of this document to restrict future changes, but
   rather to ensure such changes are easily identified and suitably
   agreed upon.

   An example tree diagram can be found in [RFC7223] Section 3. A
   portion of which follows:

```
     +--rw interfaces
     |  +--rw interface* [name]
     |     +--rw name                        string
     |     +--rw description?                string
     |     +--rw type                        identityref
     |     +--rw enabled?                    boolean
     |     +--rw link-up-down-trap-enable?   enumeration
```

2. Tree Diagram Syntax

This section provides the meaning of the symbols used in YANG Tree diagrams.

A full tree diagram of a module represents all elements. It includes the name of the module and sections for top level module statements (typically containers), augmentations, rpcs and notifications all identified under a module statement. Module trees may be included in a document as a whole, by one or more sections, or even subsets of nodes.

A module is identified by "module:" followed the module-name. This is followed by one or more sections, in order:

1. The top-level data nodes defined in the module, offset by 4 spaces.

2. Augmentations, offset by 2 spaces and identified by the keyword "augment" followed by the augment target node and a colon (":") character.

3. RPCs, offset by 2 spaces and identified by "rpcs:".

4. Notifications, offset by 2 spaces and identified by "notifications:".

5. Groupings, offset by 2 spaces, and identified by the keyword "grouping" followed by the name of the grouping and a colon (":") character.

6. yang-data, offset by 2 spaces, and identified by the keyword "yang-data" followed by the name of the yang-data structure and a colon (":") character.

The relative organization of each section is provided using a text-based format that is typical of a file system directory tree display command. Each node in the tree is prefaces with "+--". Schema nodes that are children of another node are offset from the parent by 3 spaces. Schema peer nodes separated are listed with the same space offset and, when separated by lines, linked via a vertical bar ("|") character.

The full format, including spacing conventions is:

module: <module-name>

```
      +--<node>
      |  +--<node>
      |     +--<node>
      +--<node>
         +--<node>
            +--<node>

   augment <target-node>:
     +--<node>
        +--<node>
        +--<node>
           +--<node>
   augment <target-node>:
     +--<node>

   rpcs:
     +--<rpc-node>
     +--<rpc-node>
        +--<node>
        |  +--<node>
        +--<node>

   notifications:
     +--<notification-node>
     +--<notification-node>
        +--<node>
        |  +--<node>
        +--<node>

   grouping <grouping-name>:
     +--<node>
        +--<node>
        |  +--<node>
        +--<node>
   grouping <grouping-name>:
     +--<node>

   yang-data <yang-data-name>:
     +--<node>
        +--<node>
        |  +--<node>
        +--<node>
   yang-data <yang-data-name>:
     +--<node>
```

2.1. Submodules

   Submodules are represented in the same fashion as modules, but are
   identified by "submodule:" followed the (sub)module-name. For
   example:

   submodule: <module-name>

```
     +--<node>
     |  +--<node>
     |     +--<node>
```

2.2. Groupings

   Nodes within a used grouping are expanded as if the nodes were
   defined at the location of the uses statement.

   Groupings may optionally be present in the "groupings" section.

2.3. yang-data

   If the module defines a "yang-data" structure [RFC8040], these
   structures may optionally be present in the "yang-data" section.

2.4. Collapsed Node Representation

   At times when the composition of the nodes within a module schema are
   not important in the context of the presented tree, peer nodes and
   their children can be collapsed using the notation "..." in place of
   the text lines used to represent the summarized nodes. For example:

```
     +--<node>
     |  ...
     +--<node>
        +--<node>
           +--<node>
```

2.5. Comments

   Single line comments, starting with "//" and ending at the end of the
   line, may be used in the tree notation.

2.6. Node Representation

   Each node in a YANG module is printed as:

```
   <status> <flags> <name> <opts> <type> <if-features>

     <status> is one of:

        +  for current

        x  for deprecated
        o  for obsolete

     <flags> is one of:
        rw  for configuration data
        ro  for non-configuration data
        -x  for rpcs and actions
        -n  for notifications
        mp  for nodes containing a "mount-point" extension statment

     <name> is the name of the node
        (<name>) means that the node is a choice node
       :(<name>) means that the node is a case node

        If the node is augmented into the tree from another module,
        its name is printed as <prefix>:<name>.

     <opts> is one of:
        ?  for an optional leaf, choice, anydata or anyxml
        !  for a presence container
        *  for a leaf-list or list
        [<keys>] for a list's keys
        /  for a top-level data node in a mounted module
        @  for a top-level data node in a parent referenced module

     <type> is the name of the type for leafs and leaf-lists

        If the type is a leafref, the type is printed as "-> TARGET",
        where TARGET is either the leafref path, with prefixed removed
        if possible.

     <if-features> is the list of features this node depends on,
        printed within curly brackets and a question mark "{...}?"
```

3. Usage Guidelines For RFCs

   This section provides general guidelines related to the use of tree
   diagrams in RFCs.

3.1. Wrapping Long Lines

   Internet Drafts and RFCs limit the number of characters that may in a
   line of text to 72 characters. When the tree representation of a
   node results in line being longer than this limit the line should be
   broken between <opts> and <type>. The type should be indented so
   that the new line starts below <name> with a white space offset of at
   least two characters. For example:

```
     notifications:
       +---n yang-library-change
          +--ro module-set-id
                   -> /modules-state/module-set-id
```

   The previously mentioned "pyang" command can be helpful in producing
   such output, for example the above example was produced using:

```
     pyang -f tree --tree-line-length 50 ietf-yang-library.yang
```

   When a tree diagram is included as a figure in an Internet Draft or
   RFC, "--tree-line-length 69" works well.

3.2. Long Diagrams

   As tree diagrams are intended to provide a simplified view of a
   module, diagrams longer than a page should generally be avoided. If
   the complete tree diagram for a module becomes too long, the diagram
   can be split into several smaller diagrams. For example, it might be
   possible to have one diagram with the data node and another with all
   notifications. If the data nodes tree is too long, it is also
   possible to split the diagram into smaller diagrams for different
   subtrees. When long diagrams are included in a document, authors
   should consider whether to include the long diagram in the main body
   of the document or in an appendix.

   An example of such a split can be found in [RFC7407], where section
   2.4 shows the diagram for "engine configuration":

```
         +--rw snmp
            +--rw engine
               // more parameters from the "engine" subtree here
```

   Further, section 2.5 shows the diagram for "target configuration":

```
         +--rw snmp
            +--rw target* [name]
               // more parameters from the "target" subtree here
```

The previously mentioned "pyang" command can be helpful in producing
such output, for example the above example was produced using:

```
pyang -f tree --tree-path /snmp/target ietf-snmp.yang
```

4. YANG Schema Mount Tree Diagrams

YANG Schema Mount is defined in [I-D.ietf-netmod-schema-mount] and
warrants some specific discussion. Schema mount is a generic
mechanism that allows for mounting of one or more data modules at a
specified location of another (parent) schema. The specific location
is referred to as a mount point, and any container or list node in a
schema may serve as a mount point. Mount points are identified via
the inclusion of the "mount-point" extension statement as a
substament under a container or list node. Mount point nodes are
thus directly identified in a module schema definition and can be
identified in a tree diagram as indicated above using the "mp" flag.

In the following example taken from [I-D.ietf-rtgwg-ni-model],
"vrf-root" is a container that includes the "mount-point" extension
statement as part of its definition:

```
module: ietf-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name             string
        +--rw enabled?         boolean
        +--rw description?     string
        +--rw (ni-type)?
        +--rw (root-type)
           +--:(vrf-root)
           |  +--mp vrf-root
```

4.1. Representation of Instance Data Trees

The actual modules made available under a mount point is controlled
by a server and is provided to clients. This information is
typically provided via the Schema Mount module defined in
[I-D.ietf-netmod-schema-mount]. The Schema Mount module supports
exposure of both mounted schema and "parent-references". Parent
references are used for XPath evaluation within mounted modules and
do not represent client-accessible paths; the referenced information
is available to clients via the parent schema. Schema mount also
defines an "inline" type mount point where mounted modules are
exposed via the YANG library module.

While the modules made available under a mount point are not
specified in YANG modules that include mount points, the document

defining the module will describe the intended use of the module and
may identify both modules that will be mounted and parent modules
that can be referenced by mounted modules. An example of such a
description can be found in [I-D.ietf-rtgwg-ni-model]. A specific
implementation of a module containing mount points will also support
a specific list of mounted and referenced modules. In describing
both intended use and actual implementations, it is helpful to show
how mounted modules would be instantiated and referenced under a
mount point using tree diagrams.

In such diagrams, the mount point should be treated much like a
container that uses a grouping. The flags should also be set based
on the "config" leaf mentioned above, and the mount realted options
indicated above should be shown for the top level nodes in a mounted
or referenced module. The following example, taken from
[I-D.ietf-rtgwg-ni-model], represents the prior example with YANG
Routing and OSPF modules mounted, YANG Interface module nodes
accessible via a parent-reference, and "config" indicating true:

```
module: ietf-network-instance
   +--rw network-instances
      +--rw network-instance* [name]
         +--rw name             string
         +--rw enabled?         boolean
         +--rw description?    string
         +--rw (ni-type)?
         +--rw (root-type)
            +--:(vrf-root)
               +--mp vrf-root
                  +--ro rt:routing-state/
                  |  +--ro router-id?
                  |  +--ro control-plane-protocols
                  |     +--ro control-plane-protocol* [type name]
                  |        +--ro ospf:ospf
                  |           +--ro instance* [af]
                  |           ...
                  +--rw rt:routing/
                  |  +--rw router-id?
                  |  +--rw control-plane-protocols
                  |     +--rw control-plane-protocol* [type name]
                  |        +--rw ospf:ospf
                  |           +--rw instance* [af]
                  |           ...
                  +--ro if:interfaces@
                  |  ...
                  +--ro if:interfaces-state@
                  |  ...
```

It is worth highlighting that the OSPF module augments the Routing
module, and while it is listed in the Schema Mount module (or inline
YANG library) there is no special mount-related notation in the tree
diagram.

A mount point definition alone is not sufficient to identify if the
mounted modules are used for configuration or for non-configuration
data. This is determined by the "ietf-yang-schema-mount" module's
"config" leaf associated with the specific mount point and is
indicated on the top level mounted nodes. For example in the above
tree, when the "config" for the routing module indicates false, the
only change would be to the flag on the rt:routing node:

```
                  +--ro rt:routing/
```

5. IANA Considerations

There are no IANA requests or assignments included in this document.

6. Security Considerations

There is no security impact related to the tree diagrams defined in
this document.

7. Informative References

[I-D.ietf-netmod-schema-mount]
          Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-
          ietf-netmod-schema-mount-08 (work in progress), October
          2017.

[I-D.ietf-rtgwg-ni-model]
          Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X.
          Liu, "YANG Network Instances", draft-ietf-rtgwg-ni-
          model-04 (work in progress), September 2017.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface
          Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
          <https://www.rfc-editor.org/info/rfc7223>.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for
          SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407,
          December 2014, <https://www.rfc-editor.org/info/rfc7407>.

[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>.

   [RFC8040]   Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
               Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,
               <https://www.rfc-editor.org/info/rfc8040>.

Authors' Addresses

   Martin Bjorklund
   Tail-f Systems

   Email: mbj@tail-f.com

   Lou Berger (editor)
   LabN Consulting, L.L.C.

   Email: lberger@labn.net