Network Working Group M. Bjorklund
Internet-Draft Tail-f Systems
Intended status: Standards Track L. Lhotka
Expires: September 7, 2017 CZ.NIC
March 6, 2017

YANG Schema Mount
draft-ietf-netmod-schema-mount-04

Abstract

This document defines a mechanism to combine YANG modules into the schema defined in other YANG modules.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   2
   2.  Terminology and Notation . . . . . . . . . . . . . . . . . .   5
     2.1.  Glossary of New Terms . . . . . . . . . . . . . . . . . .   6
     2.2.  Tree Diagrams . . . . . . . . . . . . . . . . . . . . . .   6
     2.3.  Namespace Prefixes . . . . . . . . . . . . . . . . . . .   6
   3.  Schema Mount . . . . . . . . . . . . . . . . . . . . . . . .   7
     3.1.  Mount Point Definition . . . . . . . . . . . . . . . . .   7
     3.2.  Specification of the Mounted Schema . . . . . . . . . . .   7
     3.3.  Multiple Levels of Schema Mount . . . . . . . . . . . . .  11
   4.  Refering to Data Nodes in the Parent Schema . . . . . . . . .  11
   5.  RPC operations and Notifications . . . . . . . . . . . . . .  12
   6.  Implementation Notes . . . . . . . . . . . . . . . . . . . .  13
   7.  Data Model . . . . . . . . . . . . . . . . . . . . . . . . .  13
   8.  Schema Mount YANG Module . . . . . . . . . . . . . . . . . .  15
   9.  IANA Considerations . . . . . . . . . . . . . . . . . . . . .  21
   10. Security Considerations . . . . . . . . . . . . . . . . . . .  21
   11. Contributors . . . . . . . . . . . . . . . . . . . . . . . .  21
   12. References . . . . . . . . . . . . . . . . . . . . . . . . .  21
     12.1.  Normative References . . . . . . . . . . . . . . . . . .  21
     12.2.  Informative References . . . . . . . . . . . . . . . . .  22
   Appendix A.  Example: Device Model with LNEs and NIs . . . . . .  23
     A.1.  Physical Device . . . . . . . . . . . . . . . . . . . . .  23
     A.2.  Logical Network Elements . . . . . . . . . . . . . . . .  24
     A.3.  Network Instances . . . . . . . . . . . . . . . . . . . .  27
     A.4.  Invoking an RPC Operation . . . . . . . . . . . . . . . .  29
   Appendix B.  Open Issues . . . . . . . . . . . . . . . . . . . .  29
     B.1.  Referencing Mount Points Using Schema Node Identifiers  .  29
     B.2.  Defining the "mount-point" Extension in a Separate Module  30
     B.3.  Parent References . . . . . . . . . . . . . . . . . . . .  31
     B.4.  RPC Operations and Notifications in Mounted Modules . . .  31
     B.5.  Tree Representation . . . . . . . . . . . . . . . . . . .  32
     B.6.  Design-Time Mounts . . . . . . . . . . . . . . . . . . .  32
   Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . .  32
```

1. Introduction

Modularity and extensibility were among the leading design principles of the YANG data modeling language. As a result, the same YANG module can be combined with various sets of other modules and thus form a data model that is tailored to meet the requirements of a specific use case. Server implementors are only required to specify all YANG modules comprising the data model (together with their revisions and other optional choices) in the YANG library data ([RFC7895], and Section 5.6.4 of [RFC7950]) implemented by the server. Such YANG modules appear in the data model "side by side", i.e., top-level data nodes of each module - if there are any - are also top-level nodes of the overall data model.

Furthermore, YANG has two mechanisms for contributing a schema hierarchy defined elsewhere to the contents of an internal node of the schema tree; these mechanisms are realized through the following YANG statements:

- The "uses" statement explicitly incorporates the contents of a grouping defined in the same or another module. See Section 4.2.6 of [RFC7950] for more details.

- The "augment" statement explicitly adds contents to a target node defined in the same or another module. See Section 4.2.8 of [RFC7950] for more details.

With both mechanisms, the source or target YANG module explicitly defines the exact location in the schema tree where the new nodes are placed.

In some cases these mechanisms are not sufficient; it is often necessary that an existing module (or a set of modules) is added to the data model starting at a non-root location. For example, YANG modules such as "ietf-interfaces" [RFC7223] are often defined so as to be used in a data model of a physical device. Now suppose we want to model a device that supports multiple logical devices [I-D.ietf-rtgwg-lne-model], each of which has its own instantiation of "ietf-interfaces", and possibly other modules, but, at the same time, we want to be able to manage all these logical devices from the master device. Hence, we would like to have a schema like this:

```
  +--rw interfaces
  |  +--rw interface* [name]
  |      ...
  +--rw logical-device* [name]
     +--rw name
     |   ...
     +--rw interfaces
       +--rw interface* [name]
           ...
```

With the "uses" approach, the complete schema tree of "ietf-interfaces" would have to be wrapped in a grouping, and then this grouping would have to be used at the top level (for the master device) and then also in the "logical-device" list (for the logical devices). This approach has several disadvantages:

- It is not scalable because every time there is a new YANG module that needs to be added to the logical device model, we have to update the model for logical devices with another "uses" statement pulling in contents of the new module.

- Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.

- Nodes defined inside a grouping belong to the namespace of the module where it is used, which makes references to such nodes from other modules difficult or even impossible.

- It would be difficult for vendors to add proprietary modules when the "uses" statements are defined in a standard module.

With the "augment" approach, "ietf-interfaces" would have to augment the "logical-device" list with all its nodes, and at the same time define all its nodes at the top level. The same hierarchy of nodes would thus have to be defined twice, which is clearly not scalable either.

This document introduces a new generic mechanism, denoted as schema mount, that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Unlike the "uses" and "augment" approaches discussed above, the mounted modules needn’t be specially prepared for mounting and, consequently, existing modules such as "ietf-interfaces" can be mounted without any modifications.

The basic idea of schema mount is to label a data node in the parent schema as the mount point, and then define a complete data model to be attached to the mount point so that the labeled data node effectively becomes the root node of the mounted data model.

In principle, the mounted schema can be specified at three different phases of the data model life cycle:

1. Design-time: the mounted schema is defined along with the mount point in the parent module. In this case, the mounted schema has to be the same for every implementation of the parent module.

2. Implementation-time: the mounted schema is defined by a server implementor and is as stable as YANG library information, i.e., it may change after an upgrade of server software but not after rebooting the server. Also, a client can learn the entire schema together with YANG library data.

3. Run-time: the mounted schema is defined by instance data that is part of the mounted data model. If there are multiple instances of the same mount point (e.g., in multiple entries of a list), the mounted data model may be different for each instance.

The schema mount mechanism defined in this document provides support only for the latter two cases because design-time definition of the mounted schema doesn’t play well with the existing YANG modularity mechanisms. For example, it would be impossible to augment the mounted data model.

Schema mount applies to the data model, and specifically does not assume anything about the source of instance data for the mounted schemas. It may be implemented using the same instrumentation as the rest of the system, or it may be implemented by querying some other system. Future specifications may define mechanisms to control or monitor the implementation of specific mount points.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms such as mounting sub-hierarchies of a module.

2. Terminology and Notation

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

The following terms are defined in [RFC6241] and are not redefined here:

- client
- notification
- server

The following terms are defined in [RFC7950] and are not redefined here:

- action
- configuration data
- container
- list

- operation

The following terms are defined in [RFC7223] and are not redefined here:

- system-controlled interface

2.1. Glossary of New Terms

- inline schema: a mounted schema whose definition is provided as part of the mounted data, using YANG library [RFC7895].

- mount point: container or list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines the name of the mount point.

- parent schema (of a particular mounted schema): the schema that contains the mount point for the mounted schema.

- top-level schema: a schema according to [RFC7950] in which schema trees of each module (except augments) start at the root node.

2.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.

- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- Ellipsis ("...") stands for contents of subtrees that are not shown.

2.3. Namespace Prefixes

In this document, names of data nodes, YANG extensions, actions and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
yangmnt	ietf-yang-schema-mount	Section 8
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895]

Table 1: Namespace Prefixes

3. Schema Mount

The schema mount mechanism defined in this document provides a new extensibility mechanism for use with YANG 1.1. In contrast to the existing mechanisms described in Section 1, schema mount defines the relationship between the source and target YANG modules outside these modules. The procedure consists of two separate steps that are described in the following subsections.

3.1. Mount Point Definition

A "container" or "list" node becomes a mount point if the "mount-point" extension (defined in the "ietf-yang-schema-mount" module) is used in its definition. This extension can appear only as a substatement of "container" and "list" statements.

The argument of the "mount-point" extension is a YANG identifier that defines the name of the mount point. A module MAY contain multiple "mount-point" statements having the same argument.

It is therefore up to the designer of the parent schema to decide about the placement of mount points. A mount point can also be made conditional by placing "if-feature" and/or "when" as substatements of the "container" or "list" statement that represents the mount point.

The "mount-point" statement MUST NOT be used in a YANG version 1 module. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

3.2. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are defined as state data in the "yangmnt:schema-mounts" container. Data in this container is intended to be as stable as data in the top-level YANG library [RFC7895]. In particular, it SHOULD NOT change during the same management session.

The "schema-mount" container has the "mount-point" list as one of its children. Every entry of this list refers through its key to a mount point and specifies the mounted schema.

If a mount point is defined in the parent schema but does not have an entry in the "mount-point" list, then the mounted schema is void, i.e., instances of that mount point MUST NOT contain any data above those that are defined in the parent schema.

If multiple mount points with the same name are defined in the same module - either directly or because the mount point is defined in a grouping and the grouping is used multiple times - then the corresponding "mount-point" entry applies equally to all such mount points.

The "config" property of mounted schema nodes is overriden and all nodes in the mounted schema are read-only ("config false") if at least one of the following conditions is satisfied for a mount point:

1. The mount point is itself defined as "config false".

2. The "config" leaf in the corresponding entry of the "mount-point" list is set to "false".

An entry of the "mount-point" list can specify the mounted schema in two different ways:

1. by stating that the schema is available inline, i.e., in run-time instance data; or

2. by referring to one or more entries of the "schema" list in the same instance of "schema-mounts".

In case 1, every instance of the mount point that exists in the parent tree MUST contain a copy of YANG library data [RFC7895] that defines the mounted schema exactly as for a top-level data model. A client is expected to retrieve this data from the instance tree, possibly after creating the mount point. Instances of the same mount point MAY use different mounted schemas.

In case 2, the mounted schema is defined by the combination of all "schema" entries referred to in the "use-schema" list. Optionally, a reference to a "schema" entry can be made conditional by including the "when" leaf. Its argument is an XPath expression that is evaluated in the parent tree with the mount point instance as the context node. The conditional "schema" entry is used only if the XPath expression evaluates to true. XPath expressions in the argument of "when" may use namespace prefixes that are declared in the "namespace" list (child of "schema-mounts").

Conditional schemas may be used, for example, in a situation where virtual devices are of several different types and the schema for each type is fixed and known in advance. The list of virtual devices in a parent schema module (say "example-virtual-host") might be defined as follows:

```
  list virtual-device {
    key name;
    leaf name {
      type string;
    }
    leaf type {
      type identityref {
        base virtual-device-type;
      }
    }
    container root {
      yangmnt:mount-point virtual-device;
    }
```

The "schema-mounts" specification in state data might contain, for example,

```
  "yangmnt:schema-mounts": {
    "namespace": [
      {
        "prefix": "evh",
        "ns-uri": "http://example.org/ns/example-virtual-host"
      }
    ],
    "mount-point": [
      {
        "module": "example-virtual-host",
        "name": "root",
        "use-schema": [
          {
            "name": "virtual-router-schema",
            "when": "derived-from(../evh:type, ’evh:virtual-router’)"
          },
          {
            "name": "virtual-switch-schema",
            "when": "derived-from(../evh:type, ’evh:virtual-switch’)"
          }
    ],
    "schema": [
      {
        "name": "virtual-router-schema",
        "module": [
          ...
        ]
      },
      {
        "name": "virtual-switch-schema",
        "module": [
          ...
        ]
      }
    ]
  }
```

The schema of virtual device instances can then be controlled by setting the "type" leaf to an appropriate identity derived from the "virtual-device-type" base.

In case 2, the mounted schema is specified as implementation-time data that can be retrieved together with YANG library data for the parent schema, i.e., even before any instances of the mount point exist. However, the mounted schema has to be the same for all instances of the mount point (except for parts that are conditional due to "when" leaves).

Each entry of the "schema" list contains

- a list in the YANG library format specifying all YANG modules (and revisions etc.) that are implemented or imported in the mounted schema;

- (optionally) a new "schema-mounts" specification that applies to mount points defined within the mounted schema.

3.3. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which subschemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of schema levels. A subschema for a mount point contained in a mounted module can be specified in one of the following ways:

- by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the subschemas exactly as it is done in the top-level schema

- by using the "mount-point" list inside the coresponding "schema" entry.

The former method is applicable to both "inline" and "use-schema" cases whereas the latter requires the "use-schema" case. On the other hand, the latter method allows for a compact representation of a multi-level schema the does not rely on the presence of any instance data.

4. Refering to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted data model works exactly as a top-level data model, i.e., it is confined to the "mount jail". This means that all paths in the mounted data model (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) are interpreted with the mount point as the root node. YANG modules of the mounted schema as well as corresponding instance data thus cannot refer to schema nodes or instance data outside the mount jail.

However, this restriction is sometimes too severe. A typical example are network instances (NI) [I-D.ietf-rtgwg-ni-model], where each NI has its own routing engine but the list of interfaces is global and shared by all NIs. If we want to model this organization with the NI schema mounted using schema mount, the overall schema tree would look schematically as follows:

```
     +--rw interfaces
     |  +--rw interface* [name]
     |     ...
     +--rw network-instances
        +--rw network-instance* [name]
           +--rw name
           +--rw root
              +--rw routing
                 ...
```

Here, the "root" node is the mount point for the NI schema. Routing configuration inside an NI often needs to refer to interfaces (at least those that are assigned to the NI), which is impossible unless such a reference can point to a node in the parent schema (interface name).

Therefore, schema mount also allows for such references, albeit in a limited and controlled way. The "schema-mounts" container has a child leaf-list named "parent-reference" that contains zero or more module names. All modules appearing in this leaf-list MUST be implemented in the parent schema and MUST NOT be implemented in the mounted schema. All absolute leafref paths and instance identifiers within the mounted data model and corresponding instance data tree are then evaluated as follows:

- If the leftmost node-identifier (right after the initial slash) belongs to the namespace of a module that is listed in "parent-reference", then the root of the accessible tree is not the mount point but the root of the parent schema.

- Other rules for the "leafref" and "instance-identifier" types as defined in Sections 9.9 and 9.13 of [RFC7950] remain in effect.

It is worth emphasizing that the mount jail can be escaped only via absolute leafref paths and instance identifiers. Relative leafref paths, "must"/"when" expressions and schema node identifiers are still restricted to the mounted schema.

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation by representing it as an action defined for the corresponding mount point, see Section 7.15 of ^RFC7950. An example of this is given in Appendix A.4.

Similarly, if the server emits a notification defined at the top level of any mounted module, it MUST be represented as if the notification was connected to the mount point, see Section 7.16 of [RFC7950].

6. Implementation Notes

Network management of devices that use a data model with schema mount can be implemented in different ways. However, the following implementations options are envisioned as typical:

- shared management: instance data of both parent and mounted schemas are accessible within the same management session.

- split management: one (master) management session has access to instance data of both parent and mounted schemas but, in addition, an extra session exists for every instance of the mount point, having access only to the mounted data tree.

7. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
   module: ietf-yang-schema-mount
       +--ro schema-mounts
          +--ro namespace* [prefix]
          |  +--ro prefix    yang:yang-identifier
          |  +--ro ns-uri?   inet:uri
          +--ro mount-point* [module name]
          |  +--ro module        yang:yang-identifier
          |  +--ro name          yang:yang-identifier
          |  +--ro config?       boolean
          |  +--ro (schema-ref)?
          |     +--:(inline)
          |     |  +--ro inline?       empty
          |     +--:(use-schema)
          |        +--ro use-schema* [name]
          |           +--ro name
          |           |       -> /schema-mounts/schema/name
          |           +--ro when?               yang:xpath1.0
          |           +--ro parent-reference*   yang:yang-identifier
          +--ro schema* [name]
             +--ro name           string
             +--ro module* [name revision]
             |  +--ro name                yang:yang-identifier
             |  +--ro revision            union
             |  +--ro schema?             inet:uri
             |  +--ro namespace           inet:uri
             |  +--ro feature*            yang:yang-identifier
             |  +--ro deviation* [name revision]
             |  |  +--ro name        yang:yang-identifier
             |  |  +--ro revision    union
             |  +--ro conformance-type    enumeration
             |  +--ro submodule* [name revision]
             |     +--ro name        yang:yang-identifier
             |     +--ro revision    union
             |     +--ro schema?     inet:uri
             +--ro mount-point* [module name]
                +--ro module        yang:yang-identifier
                +--ro name          yang:yang-identifier
                +--ro config?       boolean
                +--ro (schema-ref)?
                   +--:(inline)
                   |  +--ro inline?       empty
                   +--:(use-schema)
                      +--ro use-schema* [name]
                         +--ro name
                         |       -> /schema-mounts/schema/name
                         +--ro when?               yang:xpath1.0
                         +--ro parent-reference*   yang:yang-identifier
```

8. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

```
<CODE BEGINS> file "ietf-yang-schema-mount@2017-03-06.yang"

module ietf-yang-schema-mount {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
  prefix yangmnt;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 7895: YANG Module Library";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:   <https://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>";

  description
    "This module defines a YANG extension statement that can be used
     to incorporate data models defined in other YANG modules in a
     module. It also defines operational state data that specify the
     overall structure of the data model.
```

```
        Copyright (c) 2017 IETF Trust and the persons identified as
        authors of the code. All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject to
        the license terms contained in, the Simplified BSD License set
        forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
        'OPTIONAL' in the module text are to be interpreted as described
        in RFC 2119 (https://tools.ietf.org/html/rfc2119).

        This version of this YANG module is part of RFC XXXX
        (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
        full legal notices.";

     revision 2017-03-06 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Schema Mount";
     }

     /*
      * Extensions
      */

     extension mount-point {
       argument name;
       description
         "The argument 'name' is a YANG identifier, i.e., it is of the
          type 'yang:yang-identifier'.

          The 'mount-point' statement MUST NOT be used in a YANG
          version 1 module, neither explicitly nor via a 'uses'
          statement.

          The 'mount-point' statement MAY be present as a substatement
          of 'container' and 'list', and MUST NOT be present elsewhere.

          If a mount point is defined in a grouping, its name is bound
          to the module where the grouping is used.

          A mount point defines a place in the node hierarchy where
          other data models may be attached. A server that implements a
          module with a mount point populates the
          /schema-mounts/mount-point list with detailed information on
          which data models are mounted at each mount point.";
     }

     /*
      * Groupings
      */

     grouping mount-point-list {
       description
         "This grouping is used inside the 'schema-mounts' container and
          inside the 'schema' list.";
       list mount-point {
         key "module name";
         description
           "Each entry of this list specifies a schema for a particular
            mount point.

            Each mount point MUST be defined using the 'mount-point'
            extension in one of the modules listed in the corresponding
            YANG library instance with conformance type 'implement'. The
            corresponding YANG library instance is:

            - standard YANG library state data as defined in RFC 7895,
              if the 'mount-point' list is a child of 'schema-mounts',

            - the contents of the sibling 'yanglib:modules-state'
              container, if the 'mount-point' list is a child of
              'schema'.";
         leaf module {
           type yang:yang-identifier;
           description
             "Name of a module containing the mount point.";
         }
         leaf name {
           type yang:yang-identifier;
           description
             "Name of the mount point defined using the 'mount-point'
              extension.";
         }
         leaf config {
           type boolean;
           default "true";
           description
             "If this leaf is set to 'false', then all data nodes in the
              mounted schema are read-only (config false), regardless of
              their 'config' property.";
         }
         choice schema-ref {
           description
             "Alternatives for specifying the schema.";
           leaf inline {
             type empty;
             description
               "This leaf indicates that the server has mounted
                'ietf-yang-library' and 'ietf-schema-mount' at the mount
                point, and their instantiation (i.e., state data
                containers 'yanglib:modules-state' and 'schema-mounts')
                provides the information about the mounted schema.";
           }
           list use-schema {
             key "name";
             description
               "Each entry of this list contains a reference to a schema
                defined in the /schema-mounts/schema list. The entry can
                be made conditional by specifying an XPath expression in
                the 'when' leaf.";
             leaf name {
               type leafref {
                 path "/schema-mounts/schema/name";
               }
               description
                 "Name of the referenced schema.";
             }
             leaf when {
               type yang:xpath1.0;
               description
                 "This leaf contains an XPath expression. If it is
                  present, then the current entry applies if and only if
                  the expression evaluates to true.

                  The XPath expression is evaluated once for each
                  instance of the data node containing the mount
                  point for which the 'when' leaf is defined.

                  The XPath expression is evaluated using the rules
                  specified in sec. 6.4 of RFC 7950, with these
                  modifications:

                  - The context node is the data node instance
                    containing the corresponding 'mount-point'
                    statement.

                  - The accessible tree contains only data belonging to
                    the parent schema, i.e., all instances of data
                    nodes containing the mount points are considered
                    empty.

                  - The set of namespace declarations is the set of all
                    prefix/namespace pairs defined in the
                    /schema-mounts/namespace list. Names without a
                    namespace prefix belong to the same namespace as the
                    context node.";
             }
             leaf-list parent-reference {
               type yang:yang-identifier;
               must "not(/schema-mounts/schema[name=current()/../name]/"
                  + "module[name=current() and conformance-type="
                  + "’implement’])" {
                 error-message "Parent references cannot be used for a "
                   + "module implemented in the mounted schema.";
                 description
                   "Modules that are used for parent references MUST NOT
                    be implemented in the mounted schema.";
               }
               description
                 "Entries of this leaf-list are names of YANG modules.
                  All these modules MUST be implemented in the parent
                  schema.

                  Within the mounted schema and the corresponding data
                  tree, conceptual evaluation of absolute leafref paths
                  and instance identifiers is modified in the following
                  way:

                  If the leftmost node-identifier in an absolute leafref
                  path or instance identifier belongs to a module whose
                  name is listed in ’parent-reference’, then the root
                  of the accessible data tree coincides with the root of
                  the parent data tree.";
             }
           }
         }
       }
     }

     /*
      * State data nodes
      */

     container schema-mounts {
       config false;
       description
```

```
         "Contains information about the structure of the overall
          mounted data model implemented in the server.";
       list namespace {
         key "prefix";
         description
           "This list provides a mapping of namespace prefixes that are
            used in XPath expressions of ’when’ leafs to the
            corresponding namespace URI references.";
         leaf prefix {
           type yang:yang-identifier;
           description
             "Namespace prefix.";
         }
         leaf ns-uri {
           type inet:uri;
           description
             "Namespace URI reference.";
         }
       }
       uses mount-point-list;
       list schema {
         key "name";
         description
           "Each entry specifies a schema that can be mounted at a mount
            point.  The schema information consists of two parts:

            - an instance of YANG library that defines YANG modules used
              in the schema,

            - mount-point list with content identical to the top-level
              mount-point list (this makes the schema structure
              recursive).";
         leaf name {
           type string;
           description
             "Arbitrary name of the schema entry.";
         }
         uses yanglib:module-list;
         uses mount-point-list;
       }
     }
   }

   <CODE ENDS>
```

9. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

```
name:        ietf-yang-schema-mount
namespace:   urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
prefix:      yangmnt
reference:   RFC XXXX
```

10. Security Considerations

TBD

11. Contributors

The idea of having some way to combine schemas from different YANG modules into one has been proposed independently by several groups of people: Alexander Clemm, Jan Medved, and Eric Voit ([I-D.clemm-netmod-mount]); and Lou Berger and Christian Hopps:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>

- Alexander Clemm, Huawei, <alexander.clemm@huawei.com>

- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>

- Jan Medved, Cisco, <jmedved@cisco.com>

- Eric Voit, Cisco, <evoit@cisco.com>

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

12.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Medved, J., and E. Voit, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-05 (work in progress), September 2016.

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-15 (work in progress), February 2017.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-ietf-rtgwg-device-model-01 (work in progress), October 2016.

[I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-01 (work in progress), October 2016.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Network Instances", draft-ietf-rtgwg-ni-model-01 (work in progress), October 2016.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

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

Appendix A. Example: Device Model with LNEs and NIs

This non-normative example demonstrates an implementation of the device model as specified in Section 2 of [I-D.ietf-rtgwg-device-model], using both logical network elements (LNE) and network instances (NI).

A.1. Physical Device

The data model for the physical device may be described by this YANG library content:

```
"ietf-yang-library:modules-state": {
  "module-set-id": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2014-06-16",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-logical-network-element",
      "revision": "2016-10-21",
      "feature": [
        "bind-lne-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-library",
      "revision": "2016-06-21",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-schema-mount",
      "revision": "2017-03-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
}
```

A.2. Logical Network Elements

Each LNE can have a specific data model that is determined at run time, so it is appropriate to mount it using the "inline" method, hence the following "schema-mounts" data:

```
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "name": "root",
      "inline": [null]
    }
  ]
}
```

An administrator of the host device has to configure an entry for each LNE instance, for example,

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-logical-network-element:bind-lne-name": "eth0"
      }
    ]
  },
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "lne-1",
        "managed": true,
        "description": "LNE with NIs",
        "root": {
          ...
        }
      },
      ...
    ]
  }
}
```

and then also place necessary state data as the contents of the "root" instance, which should include at least

- YANG library data specifying the LNE’s data model, for example:

```
  "ietf-yang-library:modules-state": {
    "module-set-id": "9358e11874068c8be06562089e94a89e0a392019",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-inet-types",
        "revision": "2013-07-15",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
        "conformance-type": "import"
      },
      {
        "name": "ietf-interfaces",
        "revision": "2014-05-08",
        "feature": [
          "arbitrary-names",
          "pre-provisioning"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ip",
        "revision": "2014-06-16",
        "feature": [
          "ipv6-privacy-autoconf"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-network-instance",
        "revision": "2016-10-27",
        "feature": [
          "bind-network-instance-name"
        ],
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-network-instance",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-library",
        "revision": "2016-06-21",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-schema-mount",
        "revision": "2017-03-06",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-yang-types",
        "revision": "2013-07-15",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
      }
    ]
  }
```

o state data for interfaces assigned to the LNE instance (that effectively become system-controlled interfaces for the LNE), for example:

```
     "ietf-interfaces:interfaces-state": {
       "interface": [
         {
           "name": "eth0",
           "type": "iana-if-type:ethernetCsmacd",
           "oper-status": "up",
           "statistics": {
             "discontinuity-time": "2016-12-16T17:11:27+02:00"
           },
           "ietf-ip:ipv6": {
             "address": [
               {
                 "ip": "fe80::42a8:f0ff:fea8:24fe",
                 "origin": "link-layer",
                 "prefix-length": 64
               }
             ]
           }
         },
         ...
       ]
     }
```

A.3. Network Instances

Assuming that network instances share the same data model, it can be mounted using the "use-schema" method as follows:

```
  "ietf-yang-schema-mount:schema-mounts": {
    "mount-point": [
      {
        "module": "ietf-network-instance",
        "name": "root",
        "parent-reference": ["ietf-interfaces"],
        "use-schema": [
          {
            "name": "ni-schema"
          }
        ]
      }
    ],
    "schema": [
      {
        "name": "ni-schema",
        "module": [
          {
            "name": "ietf-ipv4-unicast-routing",
            "revision": "2016-11-04",
            "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-ipv6-unicast-routing",
            "revision": "2016-11-04",
            "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-routing",
            "revision": "2016-11-04",
            "feature": [
              "multiple-ribs",
              "router-id"
            ],
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
            "conformance-type": "implement"
          }
        ]
      }
    ]
  }
```

Note also that the "ietf-interfaces" module appears in the "parent-reference" leaf-list for the mounted NI schema. This means that references to LNE interfaces, such as "outgoing-interface" in static routes, are valid despite the fact that "ietf-interfaces" isn’t part of the NI schema.

A.4. Invoking an RPC Operation

Assume that the mounted NI data model also implements the "ietf-isis" module [I-D.ietf-isis-yang-isis-cfg]. An RPC operation defined in this module, such as "clear-adjacency", can be invoked by a client session of a LNE’s RESTCONF server as an action tied to a the mount point of a particular network instance using a request URI like this (all on one line):

```
POST /restconf/data/ietf-network-instance:network-instances/
    network-instance=rtrA/root/ietf-isis:clear-adjacency HTTP/1.1
```

Appendix B. Open Issues

B.1. Referencing Mount Points Using Schema Node Identifiers

Each entry in the "mount-point" list is currently identified by two keys, namely YANG module name and mount point name. An alternative is to use a schema node identifier of the mount point as a single key.

For example, the "schema-mounts" data for NI (Appendix A.3) would be changed as follows (the "schema" list doesn’t change):

```
"ietf-yang-schema-mount:schema-mounts": {
  "namespace": [
    {
      "prefix": "ni",
      "ns-uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
    }
  ]
  "mount-point": [
    {
      "target": "/ni:network-instances/ni:network-instance/ni:root",
      "parent-reference": ["ietf-interfaces"],
      "use-schema": [
        {
          "name": "ni-schema"
        }
      ]
    }
  ],
  "schema": [
    ...
  ]
}
```

This change would have several advantages:

- the schema mount mechanism becomes even closer to augments, which may simplify implementation

- if a mount point appears inside a grouping, then a different mounted schema can be used for each use of the grouping.

- it optionally allows for use of mount without use of the mount-point extension.

B.2. Defining the "mount-point" Extension in a Separate Module

The "inline" method of schema mounting can be further simplified by defining the "inline" case as the default. That is, if a mount point is defined through the "mount-point" extension but is not present in the "mount-point" list, the "inline" schema mount is assumed.

Consequently, a data model that uses only the "inline" method could omit the "schema-mounts" data entirely, but it still needs to use the "mount-point" extension. In order to enable this, the definition of the "mount-point" extension has to be moved to a YANG module of its own.

A variant of this approach is to completely separate the "inline" and "use-schema" cases by dedicating the "mount-point" extension for use with the "inline" method only (with no "schema-mounts" data), and using schema node identifiers as described in Appendix B.1 for the "use-schema" case.

B.3. Parent References

As explained in Section 4, references to the parent schema can only be used in absolute leafref paths and instance identifiers. However, it is conceivable that they may be useful in other XPath expressions, e.g. in "must" statements. The authors believe it is impossible to allow for parent references in general XPath expressions because, for example, in a location path "//foo:bar" it would be unclear whether the lookup has to be started in the mounted or parent schema.

Should parent references in general XPath be needed, it would be necessary to indicate it explicitly. One way to achieve this is to defining a new XPath function, e.g., parent-root(), that returns the root of the parent data tree.

B.4. RPC Operations and Notifications in Mounted Modules

Turning RPC operations defined in mounted modules into actions tied to the corresponding mount point (see Section 5, and similarly for notifications) is not possible if the path to the mount point in the parent schema contains a keyless list (Section 7.15 of [RFC7950]). The solutions for this corner case are possible:

1. any mount point MUST NOT have a keyless list among its ancestors

2. any mounted module MUST NOT contain RPC operations and/or notifications

3. specifically for each mount point, at least one of the above conditions MUST be satisfied.

4. treat such actions and notifications as non-existing, i.e., ignore them.

The first two requirements seem rather restrictive. On the other hand, the last one is difficult to guarantee - for example, things can break after an augment within the mounted schema.

B.5. Tree Representation

Need to decide how/if mount points are represented in trees.

B.6. Design-Time Mounts

The document currently doesn't provide explicit support for design-time mounts. Design-time mounts have been identified as possibly for multiple cases, and it may be worthwhile to identify a minimum or complete set of modules that must be supported under a mount point. This could be used in service modules that want to allow for configuration of device-specific information. One option could be to add an extension that specify that a certain module is required to be mounted.

Also, if design-time mounts are supported, it could be possible to represent both mounts points and their required modules in tree representations and support for such would need to be defined.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz