Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2018

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

July 5, 2017

YANG Schema Mount
draft-ietf-netmod-schema-mount-06

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology and Notation 5
2.1. Glossary of New Terms 6
2.2. Namespace Prefixes 6
3. Schema Mount 6
3.1. Mount Point Definition 7
3.2. Specification of the Mounted Schema 7
3.3. Multiple Levels of Schema Mount 9
4. Referring to Data Nodes in the Parent Schema 9
5. RPC operations and Notifications 10
6. Implementation Notes 11
7. Data Model 11
8. Schema Mount YANG Module 13
9. IANA Considerations 18
10. Security Considerations 18
11. Contributors 18
12. References 19
12.1. Normative References 19
12.2. Informative References 19
Appendix A. Example: Device Model with LNEs and NIs 20
A.1. Physical Device 21
A.2. Logical Network Elements 22
A.3. Network Instances 25
A.4. Invoking an RPC Operation 26
Authors' Addresses 27

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 YANG 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. Design-time mounts are outside the

scope of this document, and could be possibly dealt with in a future revision of the YANG data modeling language.

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

Tree diagrams used in this document follow the notation defined in [I-D.ietf-netmod-yang-tree-diagrams].

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. 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 determined from 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.

Generally, the modules that are mounted under a mount point have no relation to the modules in the parent schema; specifically, if a module is mounted it may or may not be present in the parent schema and, if present, its data will generally have no relationship to the data of the parent. Exceptions are possible and such needs to be defined in the model defining the exception, e.g., the interface module in [I-D.ietf-rtgwg-lne-model].

The "schema-mounts" 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 overridden 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:

- the mount point is itself defined as "config false"

- 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, the mounted schema is determined at run time: 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. In this case, 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. Note, that in this case a mount point may include a mounted YANG library module and the data contained in the mounted module MUST exactly match the data contained in the "schema" entries associated with the mount point.

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. Note that this includes modules that solely augment other listed modules;

- (optionally) a new "mount-point" list 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 corresponding "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. Referring 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 is 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. For every schema mounted using the "use-schema" method, it is possible to specify a leaf-list named "parent-reference" that contains zero or more XPath 1.0 expressions. Each expression is evaluated with the root of the parent data tree as the context node and the result MUST be a nodeset (see the description of the "parent-reference" node for a complete definition of the evaluation context). For the purposes of evaluating XPath expressions within the mounted data tree, the union of all such nodesets is added to the accessible data tree.

It is worth emphasizing that

- The nodes specified in "parent-reference" leaf-list are available in the mounted schema only for XPath evaluations. In particular, they cannot be accessed there via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].

- The mechanism of referencing nodes in the parent schema is not available for schemas mounted using the "inline" method.

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].

Note, inline actions and notifications will not work when they are contained within a list node without a "key" statement (see section 7.15 and 7.16 of [RFC7950]). Therefore, to be useful, mount points which contain modules with RPCs, actions, and notifications SHOULD NOT have any ancestor node that is a list node without a "key" statement. This requirement applies to the definition of modules using the "mount-point" extension statement.

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 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 parent-reference*   yang:xpath1.0
          +--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 parent-reference*   yang:xpath1.0
```


8. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

<CODE BEGINS> file "ietf-yang-schema-mount@2017-06-16.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.
```

```
      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 2017-06-16 {
    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.";
            leaf name {
              type leafref {
                path "/schema-mounts/schema/name";
              }
              description
                "Name of the referenced schema.";
            }
            leaf-list parent-reference {
              type yang:xpath1.0;
              description
                "Entries of this leaf-list are XPath 1.0 expressions
                 that are evaluated in the following context:

                  - The context node is the root node of the parent data
                    tree.

                  - The accessible tree is the parent data tree
                    *without* any nodes defined in modules that are
                    mounted inside the parent schema.

                  - The context position and context size are both equal
                    to 1.

                  - The set of variable bindings is empty.

                  - The function library is the core function library
                    defined in [XPath] and the functions defined in
                    Section 10 of [RFC7950].
```

```
                - The set of namespace declarations is defined by the
```



```
                  'namespace' list under 'schema-mounts'.

                Each XPath expression MUST evaluate to a nodeset
                (possibly empty). For the purposes of evaluating XPath
                expressions whose context nodes are defined in the
                mounted schema, the union of all these nodesets
                together with ancestor nodes are added to the
                accessible 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 'parent-reference' leafs to the
           corresponding namespace URI references.";
        leaf prefix {
          type yang:yang-identifier;
          description
            "Namespace prefix.";
        }
        leaf 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., Voit, E., and J. Medved, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-06 (work in progress), March 2017.

[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-17 (work in progress), March 2017.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-01 (work in progress), June 2017.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

[I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-03 (work in progress), July 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-03 (work in progress), July 2017.

[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>.

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

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": "2017-05-16",
        "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

o 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": "2017-05-16",
        "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": {
  "namespace": [
    {
```

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



```
          ]
        }
      ],
      "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
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz