

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 8, 2019

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC
August 7, 2018

YANG Schema Mount
draft-ietf-netmod-schema-mount-11

Abstract

This document defines a mechanism to add the schema trees defined by a set of YANG modules onto a mount point defined in the schema tree in some YANG module.

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 February 8, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Terminology and Notation	5
2.1.	Tree Diagrams	6
2.2.	Namespace Prefixes	6
3.	Schema Mount	7
3.1.	Mount Point Definition	7
3.2.	Data Model	8
3.3.	Specification of the Mounted Schema	8
3.4.	Multiple Levels of Schema Mount	9
4.	Referring to Data Nodes in the Parent Schema	9
5.	RPC operations and Notifications	10
6.	Network Management Datastore Architecture (NMDA) Considerations	11
7.	Interaction with the Network Configuration Access Control Model (NACM)	11
8.	Implementation Notes	12
9.	Schema Mount YANG Module	12
10.	IANA Considerations	17
11.	Security Considerations	17
12.	Contributors	18
13.	References	18
13.1.	Normative References	18
13.2.	Informative References	20
Appendix A.	Example: Device Model with LNEs and NIs	21
A.1.	Physical Device	21
A.2.	Logical Network Elements	23
A.3.	Network Instances	26
A.4.	Invoking an RPC Operation	27
	Authors' Addresses	28

[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], [I-D.ietf-netconf-rfc7895bis] 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.

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:

- o 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.
- o 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 YANG module with the "uses" or "augment" statement 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 sometimes necessary that an existing module (or a set of modules) is added to the data model starting at locations other than the root. For example, YANG modules such as "ietf-interfaces" [\[RFC8343\]](#) are 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 tree like this:

```
+--rw interfaces
|  +--rw interface* [name]
|  ...
+--rw logical-network-element* [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-network-element" list (for the logical devices). This approach has several disadvantages:

- o 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.

- o Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.
- o 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.
- o 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-network-element" 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 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 of the server.
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.

How and when specific mount points are instantiated by the server is out of scope for this document. Such mechanisms may be defined in future specifications.

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.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [[RFC8342](#)].

2. Terminology and Notation

The key words "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](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

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

- o action
- o container
- o data node
- o list
- o RPC operation
- o schema node
- o schema tree

The following terms are defined in [[RFC8342](#)] and are not redefined here:

- o client
- o notification
- o operational state
- o server

The following term is defined in [[RFC8343](#)] and is not redefined here:

- o system-controlled interface

The following term is defined in [[I-D.ietf-netconf-rfc7895bis](#)] is not redefined here:

- o YANG library checksum

The following additional terms are used within this document:

- o mount point: A container or a list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines a label for the mount point.
- o schema: A collection of schema trees with a common root.
- o top-level schema: A schema rooted at the root node.
- o mounted schema: A schema rooted at a mount point.
- o parent schema (of a mounted schema): A schema containing the mount point.
- o schema mount: The mechanism to combine data models defined in this document.

[2.1.](#) Tree Diagrams

Tree diagrams used in this document follow the notation defined in [[RFC8340](#)]

[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 9
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895] ,
		[I-D.ietf-netconf-rfc7895bis]

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 a label for 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 [\[RFC6020\]](#). The reason for this is that otherwise it is not possible to invoke mounted RPC operations, and receive mounted notifications. See [Section 5](#) for details. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

Note that the "mount-point" statement does not define a new data node.

3.2. 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 label]
      +--ro module          yang:yang-identifier
      +--ro label           yang:yang-identifier
      +--ro config?         boolean
      +--ro (schema-ref)
        +--:(inline)
          | +--ro inline!
        +--:(shared-schema)
          +--ro shared-schema!
            +--ro parent-reference* yang:xpath1.0
```

3.3. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are determined from state data in the "/schema-mounts" container.

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. For example, [[I-D.ietf-rtgwg-lne-model](#)] defines a mechanism to bind interfaces to mounted logical network elements.

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 except 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:

- o the mount point is itself defined as "config false"
- o 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, "inline" or "shared-schema".

The mounted schema is determined at run time: every instance of the mount point that exists in the operational state MUST contain a copy of YANG library data that defines the mounted schema exactly as for a top-level schema. A client is expected to retrieve this data from the instance tree. In the "inline" case, instances of the same mount point MAY use different mounted schemas, whereas in the "shared-schema" case, all instances MUST use the same mounted schema. This means that in the "shared-schema" case, all instances of the same mount point MUST have the same YANG library checksum. In the "inline" case, if two instances have the same YANG library checksum it is not guaranteed that the YANG library contents are equal for these instances.

3.4. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which other schemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of mounted schemas. A schema for a mount point contained in a mounted module can be specified by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the schemas exactly as it is done in the top-level schema.

4. Referring to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted schema works exactly as a top-level schema, i.e., it is confined to the "mount jail". This means that all paths in the mounted schema (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 mount point in the "shared-schema" case, 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 node in the parent data tree where the mount point is defined as the context node. The result of this evaluation 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](#)].

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation as if it were defined as an action 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. Network Management Datastore Architecture (NMDA) Considerations

The schema mount solution presented in this document is designed to work both with servers that implement the NMDA [\[RFC8342\]](#), and old servers that don't implement the NMDA.

Note to RFC Editor: please update the date YYYY-MM-DD below with the revision of the ietf-yang-library in the published version of [draft-ietf-netconf-rfc7895bis](#), and remove this note.

Specifically, a server that doesn't support the NMDA, MAY implement revision 2016-06-21 of "ietf-yang-library" [\[RFC7895\]](#) under a mount point. A server that supports the NMDA, MUST implement at least revision YYYY-MM-DD of "ietf-yang-library" [\[I-D.ietf-netconf-rfc7895bis\]](#) under the mount points.

7. Interaction with the Network Configuration Access Control Model (NACM)

If NACM [\[RFC8341\]](#) is implemented on a server, it can be used to control access to nodes defined by the mounted schema in the same way as for nodes defined by the top-level schema.

For example, suppose the module "ietf-interfaces" is mounted in the "root" container in the "logical-network-element" list defined in [\[I-D.ietf-rtgwg-lne-model\]](#). Then the following NACM path can be used to control access to the "interfaces" container (where the character '\ ' is used where a line break has been inserted for formatting reasons):


```
<path xmlns:lne=
  "urn:ietf:params:xml:ns:yang:ietf-logical-network-element"
  xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  /lne:logical-network-elements\
  /lne:logical-network-element/lne:root/if:interfaces
</path>
```

8. 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:

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

9. Schema Mount YANG Module

This module references [[RFC6991](#)].

```
<CODE BEGINS> file "ietf-yang-schema-mount@2018-08-07"

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";
  }

  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>";

// RFC Ed.: replace XXXX with actual RFC number and
// remove this note.
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) 2018 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.";

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2018-08-07 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Schema Mount";
}

/*
 * Extensions
 */
```



```
extension mount-point {
  argument label;
  description
    "The argument 'label' 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.
    There MUST NOT be more than one 'mount-point' statement in a
    given 'container' or 'list' statement.

    If a mount point is defined within a grouping, its label 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.

    Note that the 'mount-point' statement does not define a new
    data node.";
}

/*
 * 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.";
}
}
list mount-point {
  key "module label";
  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 server's
    YANG library instance with conformance type 'implement'.";
  leaf module {
    type yang:yang-identifier;
    description
      "Name of a module containing the mount point.";
  }
  leaf label {
    type yang:yang-identifier;
    description
      "Label 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 {
    mandatory true;
    description
      "Alternatives for specifying the schema.";
    container inline {
      presence
        "A complete self-contained schema is mounted at the
        mount point.";
      description
        "This node indicates that the server has mounted at least
        the module 'ietf-yang-library' at the mount point, and
        its instantiation provides the information about the
        mounted schema.
```



```
        Different instances of the mount point may have
        different schemas mounted.";
    }
    container shared-schema {
        presence
            "The mounted schema together with the 'parent-reference'
            make up the schema for this mount point.";
        description
            "This node indicates that the server has mounted at least
            the module 'ietf-yang-library' at the mount point, and
            its instantiation provides the information about the
            mounted schema. When XPath expressions in the mounted
            schema are evaluated, the 'parent-reference' leaf-list
            is taken into account.
```

```
        Different instances of the mount point MUST have the
        same schema mounted.";
    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 node in the parent data tree
              where the mount-point is defined.

            - 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.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- o /schema-mounts: The schema defined by this state data provides detailed information about a server implementation that may help an attacker identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules included in the schema, module revisions, module features, or even module deviations. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the schema information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

12. 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:

- o Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- o Alexander Clemm, Huawei, <alexander.clemm@huawei.com>
- o Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- o Jan Medved, Cisco, <jmedved@cisco.com>
- o Eric Voit, Cisco, <evoit@cisco.com>

13. References

13.1. Normative References

- [I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", [draft-ietf-netconf-rfc7895bis-06](#) (work in progress), April 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", [RFC 6020](#), DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", [RFC 6242](#), DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", [RFC 6991](#), DOI 10.17487/RFC6991, July 2013, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, [RFC 8341](#), DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", [RFC 8342](#), DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.

13.2. 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-19](#) (work in progress), November 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 Model for Logical Network Elements", [draft-ietf-rtgwg-lne-model-10](#) (work in progress), March 2018.
- [I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Network Instances", [draft-ietf-rtgwg-ni-model-12](#) (work in progress), March 2018.
- [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, <<https://www.rfc-editor.org/info/rfc6241>>.
- [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>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", [BCP 215](#), [RFC 8340](#), DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", [RFC 8343](#), DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.

[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).

In these examples, the character '\n' is used where a line break has been inserted for formatting reasons.

[A.1](#). Physical Device

The data model for the physical device may be described by this YANG library content, assuming the server supports the NMDA:

```
{
  "ietf-yang-library:yang-library": {
    "checksum": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
    "module-set": [
      {
        "name": "physical-device-modules",
        "module": [
          {
            "name": "ietf-datastores",
            "revision": "2018-02-14",
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-datastores"
          },
          {
            "name": "iana-if-type",
            "revision": "2015-06-12",
            "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type"
          },
          {
            "name": "ietf-interfaces",
            "revision": "2018-02-20",
            "feature": ["arbitrary-names", "pre-provisioning" ],
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-interfaces"
          },
          {
            "name": "ietf-ip",
            "revision": "2018-02-22",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip"
          },
          {
            "name": "ietf-logical-network-element",
            "revision": "2016-10-21",
```



```
    "feature": [ "bind-lne-name" ],
    "namespace":
      "urn:ietf:params:xml:ns:yang:\
      ietf-logical-network-element"
  },
  {
    "name": "ietf-yang-library",
    "revision": "2018-02-21",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-library"
  },
  {
    "name": "ietf-yang-schema-mount",
    "revision": "2018-03-20",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount"
  }
],
"import-only-module": [
  {
    "name": "ietf-inet-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-inet-types"
  },
  {
    "name": "ietf-yang-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-types"
  }
]
},
"schema": [
  {
    "name": "physical-device-schema",
    "module-set": [ "physical-device-modules" ]
  }
],
"datastore": [
  {
    "name": "ietf-datastores:running",
    "schema": "physical-device-schema"
  },
  {
    "name": "ietf-datastores:operational",
    "schema": "physical-device-schema"
  }
]
```



```
    }  
  ]  
}  
}
```

[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",  
        "label": "root",  
        "inline": {}  
      }  
    ]  
  }  
}
```

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, assuming the server does not implement the NMDA:

```

{
  "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": {
    "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 "shared-schema" method as follows:


```

{
  "ietf-yang-schema-mount:schema-mounts": {
    "namespace": [
      {
        "prefix": "if",
        "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
      },
      {
        "prefix": "ni",
        "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
      }
    ],
    "mount-point": [
      {
        "module": "ietf-network-instance",
        "label": "root",
        "shared-schema": {
          "parent-reference": [
            "/if:interfaces/if:interface[\
              ni:bind-network-instance-name = current()/../ni:name]"
          ]
        }
      }
    ]
  }
}

```

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