Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2016

M. Bjorklund
Tail-f Systems

February 26, 2016

YANG Structural Mount
draft-bjorklund-netmod-structural-mount-02

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

1.1. Terminology

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].

1.1.1. 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:

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

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

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

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

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

2. Background

YANG has two mechanisms for extending a data model with additional nodes; "uses" and "augment". The "uses" statement explicitly incorporates the contents of a "grouping" defined in some other module. The "augment" statement explicitly adds contents to a target node defined in some other module. In both these cases, the source and/or target model explicitly defines the relationship between the models.

In some cases these mechanisms are not sufficient. For example, suppose we have a model like ietf-interfaces [RFC7223] that is defined to be implemented in a device. Now suppose we want to model a device that supports multiple logical devices [I-D.rtgyangdt-rtgwg-device-model], where each such logical device has its own instantiation of ietf-interfaces (and other models), but at the same time, we'd like to be able to manage all these logical devices from the main device. We would like something like this:

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

With the "uses" approach, ietf-interfaces would have to define a grouping with all its nodes, and the new model for logical devices would have to use this grouping. This is a not a scalable solution, since every time there is a new model defined, we would have to update our model for logical devices to use a grouping from the new model. Another problem is that this approach cannot handle vendor-specific modules.

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 on the top-level. This approach is also not scalable, since there may be other models to which we would like to add the interface list.

3. Structural Mount

The structural mount mechanism defined in this document takes a different approach to the extensibility problem described in the previous section. It decouples the definition of the relation between the source and target models from the definitions of the models themselves.

This is accomplished with a YANG extension statement that is used to specify a mount point in a data model. The purpose of a mount point is to define a place in the node hierarchy where other YANG data models may be attached, without any special notation in the other YANG data models.

For each mount point supported by a server, the server populates an operational state node hierarchy with information about which models it has mounted. This node hierarchy can be read by a client in order to learn what is implemented on a server.

Structural mount applies to the schema, and specifically does not assume anything about how the mounted data is implemented. 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, for example mounting of sub-hierarchies of a module.

3.1. Augment and Validation in Mounted Data

All paths (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) in the data models mounted at a mount point are interpreted with the mount point as the root node, and the mounted data nodes as its children. This means that data within a mounted subtree can never refer to data outside of this subtree.

3.2. Top-level RPCs

If any mounted data model defines RPCs, these RPCs can be invoked by clients by treating them as actions defined where the mount point is specified.

3.3. Top-level Notifications

If the server emits a notification defined at the top-level in any mounted data model, it is treated as if the notification was attached to the data node where the mount point is specified.

4. Data Model

This document defines the YANG 1.1 module [I-D.ietf-netmod-rfc6020bis] "ietf-yang-structural-mount", which has the following structure:

```
   module: ietf-yang-structural-mount
      +--ro mount-points
         +--ro mount-point* [module name]
            +--ro module                 yang:yang-identifier
            +--ro name                   yang:yang-identifier
            +--ro (data-model)
               +--:(inline-yang-library)
               |  +--ro inline-yang-library?   empty
               +--:(modules)
                  +--ro modules
                     +--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 submodules
                           +--ro submodule* [name revision]
                              +--ro name        yang:yang-identifier
                              +--ro revision    union
                              +--ro schema?     inet:uri
```

5. Structural Mount YANG Module

```
   <CODE BEGINS> file "ietf-yang-structural-mount@2016-02-26.yang"

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

    import ietf-yang-types {
```

```
      prefix yang;
    }
    import ietf-yang-library {
      prefix yanglib;
    }

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

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

       WG Chair: Thomas Nadeau
                 <mailto:tnadeau@lucidvision.com>

       WG Chair: Juergen Schoenwaelder
                 <mailto:j.schoenwaelder@jacobs-university.de>

       WG Chair: Kent Watsen
                 <mailto:kwatsen@juniper.net>

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


    // 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 a operational state data so that
       clients can learn which data models a server implements for the
       mount points.

       Copyright (c) 2016 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
       (http://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 (http://tools.ietf.org/html/rfc2119).

       This version of this YANG module is part of RFC XXXX
       (http://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 2016-02-26 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Structural Mount";
    }

    /*
     * Extension statements
     */

    extension mount-point {
      argument name;
      description
        "The argument 'name' is a yang-identifier.  The name of
         the mount point MUST be unique within the module where it
         is defined.

         The 'mount-point' statement can be present in 'container' and
         'list'.

         If a mount point is defined in a grouping, its name is bound
         to the module where the grouping is used.  Note that this
         implies that such a grouping can be used at most once in a
         module.

         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
         /mount-points/mount-point list with detailed information on
         which data models are mounted at each mount point.

         The 'mount-yang-library' extension may be used as a
         substatement to 'mount-point'.";
    }

    extension mount-yang-library {
      description
        "The presence of this statement as a substatement to
         'mount-point' indicates that the data model defined in the
```

```
         module 'ietf-yang-library' is mounted.  When this statement is
         present, a client can discover the mounted YANG modules by
         reading from the mounted 'ietf-yang-library' data.

         This statement is useful if the mount point is defined in a
         list and different list entries may mount a different
         set of modules.";
    }

    /*
     * Operational state data nodes
     */

    container mount-points {
      config false;
      description
        "Contains information about which mount points are implemented
         in the server, and their data models.";

      list mount-point {
        key "module name";
        description
          "Contains information about which data models are implemented
           for the mountpoint 'name' defined in 'module'.";

        leaf module {
          type yang:yang-identifier;
          description
            "The name of the module where the mount point is defined.";
        }
        leaf name {
          type yang:yang-identifier;
          description
            "The name of the mount point.";
        }
        choice data-model {
          mandatory true;
          description
            "Indicates which data models the server implements
             for this mount point.

             It is expected that this choice may be augmented with other
             data model discovery mechansisms.";

          leaf inline-yang-library {
            type empty;
            description
              "This leaf indicates that the server has mounted
```

```
               'ietf-yang-library' at the mount point, and that the
               instantiation of 'ietf-yang-library' contains the
               information about which modules are mounted.

               This is useful if the mount point is defined in a
               list and different list entries may mount a different
               set of modules.";
          }

          container modules {
            description
              "The 'module' list contains the set of modules that are
               mounted at the mount point.";

            uses yanglib:module-list;
          }
        }
      }
    }
  }
```

   <CODE ENDS>

6. 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-structural-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-structural-mount
  namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-structural-mount
  prefix:       yangmnt
  reference:    RFC XXXX
```

7. Security Considerations

TBD

8. 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]); Ladislav Lhotka ([I-D.lhotka-netmod-ysdl]); and Lou Berger and Christian Hopps.

9. References

9.1. Normative References

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-11 (work in progress), February 2016.

[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>.

9.2. 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-03 (work in progress), April 2015.

[I-D.lhotka-netmod-ysdl]
Lhotka, L., "YANG Schema Dispatching Language", draft-lhotka-netmod-ysdl-00 (work in progress), November 2015.

[I-D.rtgyangdt-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-rtgyangdt-rtgwg-device-model-03 (work in progress), February 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>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

Appendix A. Example: Logical Devices

Logical devices within a device typically use the same set of data models in each instance. This can be modelled with a mount point:

```
module example-logical-devices {
  namespace "urn:example:logical-devices";
  prefix exld;

  import ietf-yang-structural-mount {
    prefix yangmnt;
  }

  container logical-devices {
    list logical-device {
      key name;
      leaf name {
        type string;
      }

      yangmnt:mount-point logical-device;
    }
  }
}
```

A server with two logical devices that both implement "ietf-interfaces" [RFC7223], "ietf-ip" [RFC7277], and "ietf-system" [RFC7317] YANG modules might populate the "mount-points" container with:

```
<mount-points
  xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-structural-mount">
  <mount-point>
    <module>example-logical-devices</module>
    <name>logical-device</name>
    <modules>
      <module>
        <name>ietf-interface</name>
        <revision>2014-05-08</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-interfaces
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-ip</name>
        <revision>2014-06-16</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-ip
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-system</name>
        <revision>2014-08-06</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-system
        </namespace>
        <conformance-type>implement</conformance-type>
      </module>
      <module>
        <name>ietf-yang-types</name>
        <revision>2013-07-15</revision>
        <namespace>
          urn:ietf:params:xml:ns:yang:ietf-yang-types
        </namespace>
        <conformance-type>import</conformance-type>
      </module>
    </modules>
  </mount-point>
</mount-points>
```

and the "logical-devices" container might have:

```
<logical-devices xmlns="urn:example:logical-devices">
  <logical-device>
    <name>vrtrA</name>
    <interfaces
        xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>eth0</name>
          <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <enabled>true</enabled>
            ...
          </ipv6>
        ...
      </interface>
    </interfaces>
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      ...
    </system>
  </logical-device>
  <logical-device>
    <name>vrtrB</name>
    <interfaces
        xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>eth0</name>
          <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <enabled>true</enabled>
            ...
          </ipv6>
        ...
      </interface>
    </interfaces>
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      ...
    </system>
  </logical-device>
</logical-devices>
```

Appendix B. Example: Network Manager

This example shows how a Network Manager application can use structural mount to define a data model with all its managed devices. Structural mount is used to mount the data models each device supports, and these data models can be discovered by a client via the "ietf-yang-library" module that is mounted for each device.

```
   module example-network-manager {
     namespace "urn:example:network-manager";
     prefix exnm;

     import ietf-inet-types {
       prefix inet;
     }
     import ietf-yang-structural-mount {
       prefix yangmnt;
     }

     container managed-devices {
       description
         "The managed devices and device communication settings.";

       list device {
         key name;
         leaf name {
           type string;
         }
         choice transport {
           mandatory true;
           container netconf {
             leaf address {
               type inet:ip-address;
               mandatory true;
             }
             container authentication {
               // ...
             }
           }
           container restconf {
             leaf address {
               type inet:ip-address;
               mandatory true;
             }
             // ...
           }
         }

         container root {
           yangmnt:mount-point managed-device {
             yangmnt:mount-yang-library;
           }
         }
       }
     }
   }
```

The "devices" container might have:

```
<devices xmlns="urn:example:network-manager">
  <device>
    <name>rtrA</name>
    <transport>
      <netconf>
        <address>192.0.2.2</address>
        <authentication>
          ...
        </authentication>
        ...
      </netconf>
      <root>
        <modules-state
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
          <module>
            <name>ietf-system</name>
            ...
          </module>
        </modules-state>
        <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
          ...
        </system>
      </root>
    </transport>
  </device>
  <device>
    <name>rtrB</name>
    <transport>
      <restconf>
        <address>192.0.2.3</address>
        <authentication>
          ...
        </authentication>
        ...
      </restconf>
      <root>
        <modules-state
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
          <module>
            <name>ietf-interfaces</name>
            ...
          </module>
        </modules-state>
        <interfaces
            xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          ...
```

```
        </interfaces>
      </root>
    </transport>
  </device>
</devices>
```

B.1. Invoking an RPC

A client that wants to invoke the "restart" operation [RFC7317] on the managed device "rtrA" over NETCONF [RFC6241] can send:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <managed-devices xmlns="urn:example:network-manager">
      <device>
        <name>rtrA</name>
        <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
          <restart/>
        </system>
      </device>
    </managed-devices>
  </action>
</rpc>
```

Appendix C. Open Issues

o Is there a use case for specifying modules that are required to be mounted under a mount point?

o Do we really need the case where ietf-yang-library is not mounted? The solution would be simpler if we always use ietf-yang-library at every mount point. See Appendix D.1.

o Support non-named mount points? (ysdl case) See Appendix D.2.

Appendix D. Alternative solutions

This section discusses some alternative solution ideas.

D.1. Static Mount Points with YANG Library Only

This solution supports named mount points, and always use ietf-yang-library.

There would be just one single extension statement, and no additional operational state data:

```
     extension mount-point {
       argument name;
     }
```

Data models need to be prepared with this extension:

```
     container logical-devices {
       list logical-device {
         key name;
         ...
         yangmnt:mount-point logical-device;
       }
     }
```

The tree on the server from Appendix A would look like this:

```
     "example-logical-devices:logical-devices": {
       "logical-device": [
         {
           "name": "vrtrA",
           "ietf-yang-library:modules-state": {
             "module-set-id": "ef50fe1",
             "module": [
               {
                 "name": "ietf-interfaces",
                  ...
               },
               {
                 "name": "ietf-system",
                  ...
               }
             ]
           },
           "ietf-interfaces:interfaces": {
             ...
           },
           "ietf-system:system": {
             ...
           }
         },
         {
           "name": "vrtrB",
           "ietf-yang-library:modules-state": {
             ...
           }
         }
       ]
     }
```

D.2. Dynamic Mount Points with YANG Library Only

This solution supports only non-named mount points, and always use ietf-yang-library.

There would be no extension statement. Instead, the server would populate a list of dynamic mount points. Each such mount point MUST mount ietf-yang-library.

```
  container mount-points {
    config false;
    list mount-point {
      key path;
      leaf path {
        type schema-node-path;
      }
    }
  }
```

The tree on the server from Appendix A would look like this:

```
     "ietf-yang-structural-mount:mount-points": {
       "mount-point": [
         { "path": "/exld:logical-devices/exld:logical-device" }
       ]
     },
     "example-logical-devices:logical-devices": {
       "logical-device": [
         {
           "name": "vrtrA",
           "ietf-yang-library:modules-state": {
             "module-set-id": "ef50fe1",
             "module": [
               {
                 "name": "ietf-interfaces",
                  ...
               },
               {
                 "name": "ietf-system",
                  ...
               }
             ]
           },
           "ietf-interfaces:interfaces": {
             ...
           },
           "ietf-system:system": {
             ...
           }
         },
         {
           "name": "vrtrB",
           "ietf-yang-library:modules-state": {
             ...
           }
         }
       ]
     }
```

A client needs to read the "/mount-points/mount-point" list in order to learn where the server has mounted data models. Next, it needs to read the "modules-state" subtree for each instantiated mount point in order to learn which modules are mounted at that instance.

Author's Address

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com