
YANG Schema Mount

draft-ietf-schema-mount-03

Martin Björklund

<mbj@tail-f.com>

Ladislav Lhotka

<lhotka@nic.cz>

17 November 2016

Main Changes since -02


- Any number of mount levels can be specified in place.
- Mounted schemas can be defined as conditional, based on an XPath expression evaluated on the *parent* tree.
- The `mount-yang-library` extension is not used.

Mount Points

Top-level modules (+revisions, submodules, features and deviations) are specified using *YANG library* [RFC 7895].

Mount points are defined inside these modules using the `mount-point` extension defined in *ietf-yang-schema-mount* – only under **anydata**!

```
import ietf-yang-schema-mount {  
    prefix yangmnt;  
}  
    // anywhere in the schema tree  
    anydata foo {  
        yangmnt:mount-point foo-mount-point;  
    }  
...  
}
```



arbitrary name

Each mount point is identified by its name and module name.

For each mount point, the schema is specified using state data defined in *ietf-yang-schema-mount*.

Mounted Schemas

```
list mount-point {  
  key "module name";  
  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."  
  }  
  choice subschema-ref {  
    leaf inline {  
      type empty;  
    }  
    list use-schema {  
      ...  
    }  
  }  
}
```

 redirection to state data under the mount point

Schema Reference


```
list use-schema {  
  key "name";  
  leaf name {  
    type leafref {  
      path "/schema-mounts/schema/name";  
    }  
  }  
  leaf when {  
    type yang:xpath1.0;  
  }  
}
```

If when is present, the schema entry is used only if the XPath expression evaluates to true.

Context for XPath evaluation:

- context node: **anydata** with the mount point
- accessible tree: only parent tree
- function library: as in YANG 1.1
- namespace declarations: explicitly specified

```
list namespace {  
  key "prefix";  
  leaf prefix {  
    type yang:yang-identifier;  
  }  
  leaf ns-uri {  
    type inet:uri;  
  }  
}
```



Schema Specification

```
list schema {  
  key "name";  
  leaf name {  
    type string;  
    description  
      "Arbitrary name of the entry.";  
  }  
  uses yanglib:module-list; ← module list for this schema  
  uses mount-point-list; ← mount points for this schema  
}
```

The inner mount point list refers to mount points in modules specified in the inner YANG library.

Complete Schema Tree (Part 1)

```
module: ietf-yang-schema-mount
  +--ro schema-mounts
    +--ro namespace* [prefix]
      | +--ro prefix yang:yang-identifier
      | +--ro ns-uri? inet:uri
    +--ro mount-point* [module name]
      | +--ro module yang:yang-identifier
      | +--ro name yang:yang-identifier
      | +--ro (subschema-ref)?
      |   +--:(inline)
      |   | +--ro inline? empty
      |   +--:(use-schema)
      |     +--ro use-schema* [name]
      |       +--ro name -> /schema-mounts/schema/name
      |       +--ro when? yang:xpath1.0
```

Complete Schema Tree (Part 2)

```
+--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 (subschema-ref)?
  +--:(inline)
  | +--ro inline?      empty
  +--:(use-schema)
  +--ro use-schema* [name]
  +--ro name          -> /schema-mounts/schema/name
  +--ro when?         yang:xpath1.0
```


Use Case #1

Arbitrary mounted schemas, not known in advance – each device has a specific schema.

```
module example-network-manager {  
  ...  
  list device {  
    key name;  
    leaf name {  
      type string;  
    }  
    anydata device-root {  
      yangmnt:mount-point managed-device;  
    }  
  }  
}
```

```
"ietf-yang-schema-mount:schema-mounts": {  
  "mount-point": [  
    {  
      "module": "example-network-manager",  
      "name": "managed-device",  
      "inline": [null]  
    }  
  ]  
}
```

An instance of YANG library (and schema-mounts) must be present under device-root in every entry of device list.

Use Case #2

Alternative schemas, all known in advance – many devices sharing a few schemas.

```
module example-network-manager {
  ...
  list device {
    key name;
    leaf name {
      type string;
    }
    leaf type {
      type identityref {
        base edt:device-type;
      }
    }
    anydata device-root {
      yangmnt:mount-point managed-device;
    }
  }
}
```

Use Case #2, continued

```
"ietf-yang-schema-mount:schema-mounts": {  
  "namespace": [  
    { "prefix": "edt",  
      "ns-uri": "http://example.org/device-types" }  
  ],  
  "mount-point": [  
    { "module": "example-network-manager",  
      "name": "managed-device",  
      "use-schema": [  
        { "name": "switch",  
          "when": "derived-from-or-self(..../type, 'edt:switch')" },  
        { "name": "router",  
          "when": "derived-from-or-self(..../type, 'edt:router')" },  
      ]  
    }  
  ]  
}
```

XPath expressions in when are evaluated with device-root as the context node.

Open Issues

1. Is the mount-point extension really needed?
2. Is it possible to handle “hybrid” mounts?

Extension mount - point

Pros:

- Data modeller's intention is expressed in a machine readable form.

Cons:

- If a mount point is defined inside a grouping, then the grouping can be used no more than once in the same module.

Alternative: Use schema node identifiers for locating mount points (still limited to anydata nodes).

“Hybrid” Mounts

Currently all mounted schemas are self-contained and cannot refer to ancestor or descendant schemas.

- A. Routing instances: mounted schema (*ietf-routing*, ...), but all instances share a global list of interfaces (*ietf-interfaces*, ...).
- B. Virtual routers: Physical device has a global list of interfaces, but each virtual router also has its own list of interfaces, along with routing configuration.

Even if we invent notation for referring to nodes in ancestor schemas, a module such as *ietf-routing* cannot support both variants at the same time.

Possible solution: keep an extra list of interfaces allocated to each routing instance, and specify this allocation in the global interface list.

(Déjà vu: draft-ietf-netmod-routing-cfg-19).