

---

# **YANG Schema Dispatching Language**

draft-lhotka-netmod-ysdl-00

Ladislav Lhotka  
<lhotka@nic.cz>

22 February 2016

---

# The Problem

It is relatively easy to reuse peripheral parts of YANG data trees:

- Their content is dictated by specific technologies and/or protocols.
- Variability due to optional functions or vendor additions can be handled by the existing YANG mechanisms: **module**, **feature** (and **deviation**).

Different (classes of) devices often differ in top levels of the data hierarchy, mainly due to various forms of device virtualisation. This variability cannot be accommodated without rewriting existing YANG modules.

*Reason:* YANG modules comprising a data model have a flat organisation, each starts from the root node of the data tree (except augments).

# The Solution (tl;dr)

YSDL essentially defines *external augments*: a schema tree is grafted to a schema node in another module.

The result is the similar to wrapping the modules' content in an **augment** statement with the same target node.

The details that need to be worked out are relatively straightforward.

# YSDL Characteristics

- No really new data-modelling mechanism: it works like an augment.
- Static schema: the set of modules is fixed (and advertised via *yang-library*), no run-time additions. The complete schema can be retrieved from one place.
- Backward compatible: single yang-library with the same restrictions, the default meta-schema is the one that's used by current servers (modules are organised side by side).
- No changes to YANG, no extensions: target nodes for regular augments are not tagged as such, why would we need to specify "mount points" here?

# Details

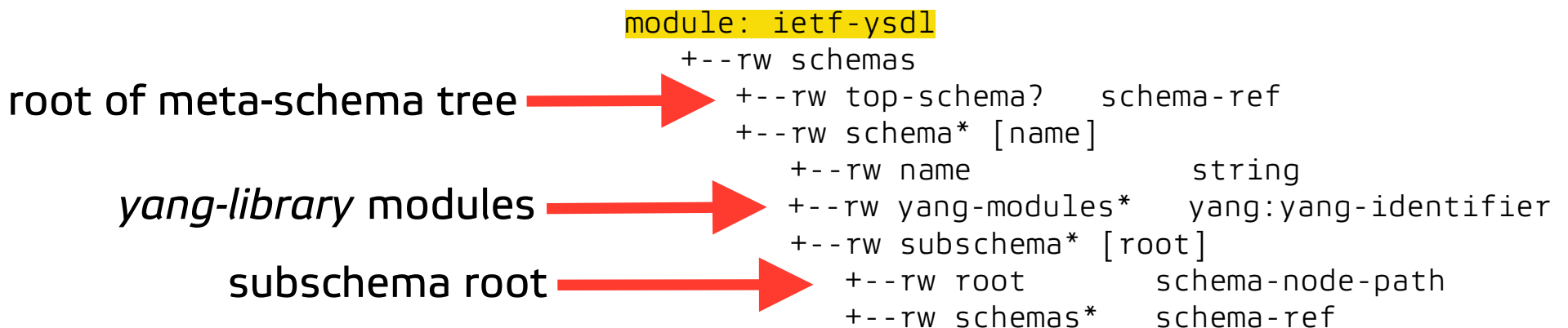
Schema:

- list of modules from *yang-library* (conformance-type=implemented)
- list of subschemas

Each subschema has its root – internal schema node, including **case** – in one of the schema's modules.

Each schema must be self-contained.


Exactly one schema must be designated as the top-level schema.



# Example

```
{
  "ietf-ysdl:schemas": {
    "top-schema": "device",
    "schema": [
      {
        "name": "device",
        "yang-modules": [
          "network-device"
        ],
        "subschema": [
          ...
        ]
      },
      {
        "name": "if-ip",
        "yang-modules": [
          "ietf-interfaces",
          "ietf-ip"
        ]
      },
      {
        "name": "system",
        "yang-modules": [
          "ietf-system"
        ]
      }
    ]
  }
}
```

**module: example-device**  
+--rw device  
+--rw logical-NEs  
+--rw logical-NE\* [name]  
+--rw name string

 see next slide

# Example (continued)

```
"subschema": [  
  {  
    "root": "network-device:device",  
    "schemas": [  
      "if-ip",  
      "system"  
    ]  
  },  
  {  
    "root": "network-device:device/logical-network-element",  
    "schemas": [  
      "if-ip"  
    ]  
  }  
]
```

# Example – Resulting Schema

```
module: network-device
```

```
+--rw device
  +--rw if:interfaces
  |   ...
  +--ro if:interfaces-state
  |   ...
  +--rw sys:system
  |   ...
  +--rw sys:system-state
  |   ...
  +--rw logical-network-element [name]
    +--rw name      string
    +--rw if:interfaces
    |   ...
    +--ro if:interfaces-state
    |   ...
```



# To Do

- ❶ Restrictions on schema recursion.
- ❷ Implement YSDL as a server resource:
  - integrate YSDL into *yang-library*,
  - provide it as a separate resource.
- ❸ Applicability of RPCs in subschemas:

similar solution as in *structural-mount* – turn RPCs into actions bound to the root node or its closest ancestor **container** or **list**.