Overview and Objectives

This presentation uses an example to walk through all main features in the YANG data modeling language.

Our example uses standard and draft-standard YANG modules for static MPLS LSPs with the goal of creating valid configuration.

After this presentation, you should be able to:

- Identify and describe common elements of a YANG model
- Examine a YANG model and create a valid configuration instance
YANG Models Used

IETF Standard Track YANG Models:
- RFC 6991  Common YANG Data Types
- RFC 7277  IP Management
- RFC 7224  IANA Interface Type
- RFC 7223  Interface Management

Draft YANG Models:
- OpenConfig MPLS LSP Model

Feel free to download and follow!
Our Use Case – MPLS VPN Configuration

Tasks:
1. Enable interfaces on routers
2. Assign IPv6 addresses to interfaces
3. Configure Static MPLS LSPs

Router 1:
eth0: 2001:db8:c18:1::3/128

Router 2:
eth0: 2001:db8:c18:1::2/128
Task #1: Enabling the Interfaces

• We start with the Interface Management Model
• Examine model for YANG features:
  – Structure
  – Configuration and operational data
  – Built-in and customer data types
  – Conditional features
  – Abstract identities
  – Nodes references
Two top-level containers:

**interfaces**
- One entry per configured interface
- Contains all configuration per interface

**interfaces-state**
- One entry per interface on the device
- Contains all operational state per interface
The Interfaces List

The Model

Example Instance data
The Module Header

RFC 7223 Interface Management

```yang
module ietf-interfaces {
    namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
    prefix if;
    import ietf-yang-types {
        prefix yang;
    }

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

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

    description
        "This module contains a collection of YANG definitions for
        managing network interfaces."

    revision 2014-05-08 {
        description
            "Initial revision."
        reference
            "RFC 7223: A YANG Data Model for Interface Management"
    }
}
```
Defining a Container

Container statement:
- Defines an interior data node in the schema tree
- One argument - identifier
- No value, but has a list of child nodes in the data tree

```
container interfaces {
    description
        "Interface configuration parameters."
    ...
}
```
Defining a List

List statement:
- Defines an interior data node in the schema tree.
- Single argument - identifier,
- Represents a collection of entries – each entry consists of one or more nodes

```
container interfaces{
    ...
    list interfaces {
        key "name";
        description
            "The list of configured
            interfaces on the device."
    ...
    }
    list interfaces-state{
        config false;
        key "name";
        description
            "Data nodes for the operational
            state of interfaces."
    ...
    }
}
```

**config false** – Data under `interfaces-state` is read-only

Config (RW) and State (RO) clearly separated in this model
Defining Leaves

A leaf is defined by an identifier and has a type

```go
list interface{
    key "name";
    description "...";

    leaf name {
        type string;
        description
            "The name of the interface"
    }
}
```

- **Leaf name** serves as list key
- The type is string

The Model

```
interface

name string

identifier

name
```

An Instance of the Model

```
interface

"eth0"

"Eth1"

"eth0"
```

Switch order on graphic
YANG Data Types

YANG has a set of built-in types, similar to those of many programming languages:

- binary
- bits
- boolean
- decimal64
- empty
- enumeration
- identityref
- instance-identifier
- int8, int16, int32, int64
- leafref
- string
- uint8, uint16, uint32, uint64
- union

Use `pattern`, `range`, and `length` statements to restrict values:

```yml
type string {
    length "0..4";
    pattern "[0-9a-fA-F]*";
}
```
Leaf Types - Boolean

Leaf enabled with boolean value true or false
This is where the interface can be enabled and disabled

```yaml
list interface{
  key "name";
  description "...";
  leaf name {...}

  leaf enabled {
    type boolean;
    default "true";
    description "This leaf contains the configured, desired state of the interface."
  }
}
```

The Model

An Instance of the Model
Leaf Types - Enumeration

Leaf \texttt{link-up-down-trap-enable} may take value \texttt{enabled} or \texttt{disabled}

```yaml
list interface{
  key "name";
  leaf name {...}
  leaf enabled {...}

  leaf link-up-down-trap-enable {
    if-feature if-mib;
    type enumeration {
      enum enabled {value 1;}
      enum disabled {value 2;}
    }
    description
    "Controls whether linkUp/Down SNMP notifications should be generated"
  }
}
```

The Model

- **name**: string (RW)
- **enabled**: boolean
- **link-up-down-trap-enable**: enumeration

Switch order on graphic
Defining new types

New types can be defined using the `typedef` statement

```yml
typedef percent {
    type uint8 {
        range "0 .. 100";
    }
    description "Percentage";
}
leaf completed {
    type percent;
}
```

RFC 6991: Common YANG Data Types
- ietf-inet-types (ipv4- and ipv6-addresses, domain-name, etc)
- ietf-yang-types (counters, gauges, date-and-time, etc)
Conditional Leaves - Features

The `feature` statement is used to mark parts of the model as conditional.
The `if-feature` statement makes the parent statement conditional.

This leaf is a part of our model only if the `if-mib` feature is supported in the server.
Abstract Types - Identityref

identity interface-type {
  description
  "Base identity from which specific interface types are derived."
}

leaf type {
  type identityref {
    base interface-type;
  }
  mandatory true;
  description
  "The type of the interface...."
  reference
  "RFC 2863: The Interfaces Group MIB - ifType"
}
The Instance Data in XML

Router 1: eth0: 2001:db8:c18:1::3/128
Router 2: eth0: 2001:db8:c18:1::2/128

XML Representation:

```
<interfaces>
  <interface>
    <name>eth0</name>
    <enabled>true</enabled>
  </interface>
</interfaces>
```
Inspecting the Interfaces State Tree

Each entry in the interfaces-state list is a container representing the state of an interface.

```
+-ro interfaces-state
  +--ro interface* [name]
     |     +--ro name string
     |     +--ro type identityref
     |     +--ro admin-status enumeration
     |     +--ro oper-status enumeration
     |     +--ro last-change? yang:date-and-time
     |     +--ro if-index int32
     |     +--ro phys-address? yang:phys-address
     |     +--ro higher-layer-if* interface-state-ref
     |     +--ro lower-layer-if* interface-state-ref
     +--ro speed? yang:gauge64
     +--ro statistics
        |     +--ro discontinuity-time yang:date-and-time
        |     +--ro in-octets? yang:counter64
        |     +--ro in-unicast-pkts? yang:counter64
        |     +--ro in-broadcast-pkts? yang:counter64
        |     +--ro in-multicast-pkts? yang:counter64
        |     +--ro in-discards? yang:counter32
        |     +--ro in-errors? yang:counter32
        |     +--ro in-unknown-protos? yang:counter32
        ...
Imports and Includes

```plaintext
... import ietf-yang-types {
    prefix yang;
}
...
```

YANG structures data models into modules and submodules.

- The `import` statement makes definitions from one module available inside another module or submodule.
- The `include` statement is used to make content from a submodule available to that submodule’s parent module, or to another submodule of that parent module.
Example Import

import ietf-yang-types {
    prefix yang;
}

leaf last-change {
    type yang:date-and-time;
    description "The time the interface entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this node is not present.";
    reference "RFC 2863: The Interfaces Group MIB - ifLastChange";
}

typedef date-and-time {
    type string {
        pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?' + '(Z|\[\+\-]\d{2}:\d{2})';
    }
    description "The date-and-time type is a profile of the ISO 8601 standard for representation of dates and times using the …";
}
Example derived type

defined type
date-and-time {
    type string {
        pattern '\d{4}\d{2}\d{2}T\d{2}:\d{2}:\d{2}[^\d]+(\.[0-9]+)?\+\[\+\-\]?:\d{2}'
    }
}
Referencing Another Leaf - leafref

Use `leafref` to reference a particular leaf instance in the data tree.

typedef `interface-state-ref` {
    type `leafref` {
        path "'/if:interfaces-state/if:interface/if:name';"
    }
    ...
}

`leaf-list lower-layer-if` {
    type `interface-state-ref`;
    ...
}
Task #2: Assigning an IPv6 Address

To assign IP address we need to augment `ietf.interfaces.yang` with `ietf-ip.yang`
Augmenting the Interface Definition

ietf-ip.yang

```yaml
import ietf-interfaces {
  prefix if;
}

augment "/if:interfaces/if:interface" {
  description
    "Parameters for configuring IP on interfaces...";
  container ipv6 {
    presence
      "Enables IPv6 unless the 'enabled' leaf (which defaults to 'true') is set to 'false'";
    description
      "Parameters for the IPv6 address family.";
  }
}
```
The IPv6 Model

container ipv6 {  
  leaf enabled {  
    type boolean;  
    default true;  
    description "Controls whether IPv6 is enabled or disabled...";  
  }  
  list address {  
    key "ip";  
    description "list of configured IPv6 addresses on interface.";  
    leaf ip {  
      type inet:ipv6-address-no-zone;  
      description "The IPv6 address on the interface.";  
    }  
    leaf prefix-length {  
      type uint8 { range "0..128"; }  
      mandatory true;  
      description "The length of the subnet prefix.";  
    }  
  }  
}
The Instance Data in XML

```
<interfaces>
  <interface>
    <name>eth0</name>
    <enabled>true</enabled>
    <ipv6>
      <enabled>true</enabled>
      <address>
        <ip>2001:0db8:c18:1::2</ip>
        <prefix-length>128</prefix-length>
      </address>
    </ipv6>
  </interface>
</interfaces>
```

Router #1

```
<interfaces>
  <interface>
    <name>eth0</name>
    <enabled>true</enabled>
    <ipv6>
      <enabled>true</enabled>
      <address>
        <ip>2001:0db8:c18:1::3</ip>
        <prefix-length>128</prefix-length>
      </address>
    </ipv6>
  </interface>
</interfaces>
```

Router #2
Task #3: Configure an LSP
Task #3: Configure LSPs

Container static-lsps will hold our configuration

Please note the uses statement below

```
container mpls {
    presence "top-level container for MPLS config and state";
    ...
    container lsps {
        description "LSP definitions and configuration";
        container static-lsps {
            description "statically configured LSPs, without dynamic signaling";
            uses static-lsp-main;
        };
    }
}
```
Groupings

Groups of nodes can be assembled into reusable collections using the grouping statement.

A grouping defines a set of nodes that are instantiated with the uses statement:

grouping static-lsp-main {
  list label-switched-path {
    key name;
    leaf name { type string; }
    container ingress { uses static-lsp-common; }
    container egress { uses static-lsp-common; }
  }
}

grouping static-lsp-common {
  leaf next-hop { type inet:ip-address; }
  leaf incoming-label { type mplst:mpls-label; }
  leaf push-label { type mplst:mpls-label; }
}
The Instance Data in XML

Router #1

```xml
<mpls>
  <lsps>
    <static-lsps>
      <label-switched-path>
        <name>lsp0</name>
        <ingress>
          <incoming-label>100</incoming-label>
        </ingress>
        <egress>
          <next-hop>2001:db8:c18:1::2</next-hop>
        </egress>
      </label-switched-path>
    </static-lsps>
  </lsps>
</mpls>
```

Router #2

```xml
<mpls>
  <lsps>
    <static-lsps>
      <label-switched-path>
        <name>lsp0</name>
        <ingress>
          <incoming-label>100</incoming-label>
        </ingress>
        <egress>
          <next-hop>2001:db8:c18:1::3</next-hop>
        </egress>
      </label-switched-path>
    </static-lsps>
  </lsps>
</mpls>
```
Summary

You should now be able to:

- Identify and describe common elements of a YANG model
- Examine a YANG model and create a valid configuration instance
Back Matter

• This material was originally developed by Charlie Justus and Carl Moberg with the support of Cisco Systems, special thanks to:
  – Kevin Serveau
Changelog

• 1.0 (2015-10-05) – Initial version
  Carl Moberg <camoberg@cisco.com>