YANG Models

A YANG model defines a data model through the data of the router, and the hierarchical organization and constraints on that data. Each module is uniquely identified by a namespace URL. The YANG models describe the configuration and operational data, perform actions, remote procedure calls, and notifications for network devices.

The YANG models must be obtained from the router. The models define a valid structure for the data that is exchanged between the router and the client. The models are used by NETCONF and gRPC-enabled applications.

YANG models can be:

- **Cisco-specific models:** For a list of supported models and their representation, see https://github.com/YangModels/yang/tree/master/vendor/cisco/svo.

- **Common models:** These models are industry-wide standard YANG models from standard bodies, such as IETF and IEEE. These models are also called Open Config (OC) models. Like synthesized models, the OC models have separate YANG models defined for configuration data and operational data, and actions.

For a list of supported OC models and their representation, see https://github.com/YangModels/yang/tree/master/vendor/cisco/svo.

Components of a YANG Model

A YANG model defines a single data model. However, a module can reference definitions in other modules and sub-modules by using one of these statements:

- **import** imports external modules

- **include** includes one or more sub-modules

- **augment** provides augmentations to another module, and defines the placement of new nodes in the data model hierarchy

- **when** defines conditions under which new nodes are valid

• **prefix** references definitions in an imported module

The YANG models configure a feature, retrieve the operational state of the router, and perform actions.

Structure of YANG Models

YANG data models can be represented in a hierarchical, tree-based structure with nodes. This representation makes the models easy to understand.

Each feature has a defined YANG model, which is synthesized from schemas. A model in a tree format includes:

• Top level nodes and their subtrees

• Subtrees that augment nodes in other YANG models

• Custom RPCs

YANG defines four node types. Each node has a name. Depending on the node type, the node either defines a value or contains a set of child nodes. The nodes types for data modeling are:

• leaf node - contains a single value of a specific type

• leaf-list node - contains a sequence of leaf nodes

• list node - contains a sequence of leaf-list entries, each of which is uniquely identified by one or more key leaves

• container node - contains a grouping of related nodes that have only child nodes, which can be any of the four node types

Communication Protocols

Communication protocols establish connections between the router and the client. The protocols help the client to consume the YANG data models to, in turn, automate and programme network operations.

YANG uses the Network Configuration Protocol (NETCONF) protocol.

The transport and encoding mechanisms for this protocol is shown in the table:

Protocol	Transport	Encoding/ Decoding
NETCONF	ssh	xml

NETCONF Protocol

NETCONF provides mechanisms to install, manipulate, or delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. Use **ssh server capability netconf-xml** command to enable NETCONF to reach XML subsystem via port 22. NETCONF uses a simple RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. The client can be a script or application that runs as part of a network manager. The server is a network device such as a router.

NETCONF Session

A NETCONF session is the logical connection between a network configuration application (client) and a network device (router). The configuration attributes can be changed during any authorized session; the effects are visible in all sessions. NETCONF is connection-oriented, with SSH as the underlying transport. NETCONF sessions are established with a "hello" message, where features and capabilities are announced. Sessions are terminated using *close* or *kill* messages.

NETCONF Layers

NETCONF can be partitioned into four layers:

Figure 1: NETCONF Layers
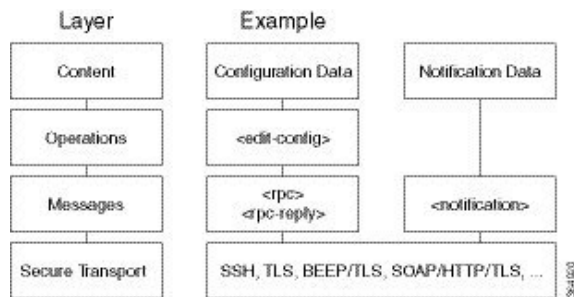
- **Content layer:** includes configuration and notification data

- **Operations layer:** defines a set of base protocol operations invoked as RPC methods with XML-encoded parameters

- **Messages layer:** provides a simple, transport-independent framing mechanism for encoding RPCs and notifications

- **Secure Transport layer:** provides a communication path between the client and the server

NETCONF Operations

NETCONF defines one or more configuration datastores and allows configuration operations on the datastores. A configuration datastore is a complete set of configuration data that is required to get a device from its initial default state into a desired operational state. The configuration datastore does not include state data or executive commands.

The base protocol includes the following NETCONF operations:

```
|   +--Get-config
|   +--Edit-Config
|       +--Merge
|       +--Replace
|       +--Create
|       +--Delete
|       +--Remove
|       +--Default-Operations
|           +--Merge
|           +--Replace
|           +--None
|   +--Get
|   +--Lock
|   +--UnLock
```

```
|   +--Close-Session
|   +--Kill-Session
```

NETCONF Operation	Description	Example
\<get-config\>	Retrieves all or part of a specified configuration from a named data store	Retrieve specific interface configuration details from running configuration using filter option ``` <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <get-config> <source> <running/> </source> <filter> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"\> <interface-configuration> <active>act</active> <interface-name>TenGigE0/0/0/2/0</interface-name> </interface-configuration> </interface-configurations> </filter> </get-config> </rpc> ```
\<get\>	Retrieves running configuration and device state information	Retrieve all acl configuration and device state information. ``` Request: <get> <filter> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/> </filter> </get> ```

NETCONF Operation	Description	Example
\<edit-config\>	Loads all or part of a specified configuration to the specified target configuration	Configure ACL configs using **Merge** operation ``` <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target><candidate/></target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-cfg" xc:operation="merge"> <accesses> <access> <access-list-name>aclv4-1</access-list-name> <access-list-entries> <access-list-entry> <sequence-number>10</sequence-number> <remark>GUEST</remark> </access-list-entry> <access-list-entry> <sequence-number>20</sequence-number> <grant>permit</grant> <source-network> <source-address>172.0.0.0</source-address> <source-wild-card-bits>0.0.255.255</source-wild-card-bits> </source-network> </access-list-entry> </access-list-entries> </access> </accesses> </ipv4-acl-and-prefix-list> </config> </edit-config> </rpc> Commit: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <commit/> </rpc> ```
\<lock\>	Allows the client to lock the entire configuration datastore system of a device	Lock the running configuration. ``` Request: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <lock> <target> <running/> </target> </lock> </rpc> Response : <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply> ```

NETCONF Operation	Description	Example
| <Unlock> | Releases a previously locked configuration.<br><br>An <unlock> operation will not succeed if either of the following conditions is true:<br><br>• The specified lock is not currently active.<br><br>• The session issuing the <unlock> operation is not the same session that obtained the lock. | Lock and unlock the running configuration from the same session.<br><br>```
Request:
rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<unlock>
<target>
<running/>
</target>
</unlock>
</rpc>

Response -
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
``` |
| <close-session> | Closes the session. The server releases any locks and resources associated with the session and closes any associated connections. | Close a NETCONF session.<br><br>```
Request :
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<close-session/>
 </rpc>

Response:
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
``` |
| <kill-session> | Terminates operations currently in process, releases locks and resources associated with the session, and close any associated connections. | Terminate a session if the ID is other session ID.<br><br>```
Request:
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<kill-session>
 <session-id>4</session-id>
 </kill-session>
 </rpc>

Response:
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
``` |