Data models

Data modeling is standard based approach to model configuration and operational data in networking devices. Using data models, customers can automate and simplify network wide visibility and configuration.

DataModels-ProgrammaticandStandards-basedConfiguration

Cisco IOS XR software supports the automation of configuration of multiple routers across the network using Data models. Configuring routers using data models overcomes drawbacks posed by traditional router management techniques.

CLIs are widely used for configuring a router and for obtaining router statistics. Other actions on the router, such as, switch-over, reload, process restart are also CLI-based. Although, CLIs are heavily used, they have many restrictions.

Customer needs are fast evolving. Typically, a network center is a heterogenous mix of various devices at multiple layers of the network. Bulk and automatic configurations need to be accomplished. CLI scraping is not flexible and optimal. Re-writing scripts many times, even for small configuration changes is cumbersome. Bulk configuration changes through CLIs are error-prone and may cause system issues. The solution lies in using data models - a programmatic and standards-based way of writing configurations to any network device, replacing the process of manual configuration. Data models are written in a standard, industry-defined language. Although configurations using CLIs are easier (more human-friendly), automating the configuration using data models results in scalability.

Cisco IOS XR supports the YANG data modeling language. YANG can be used with Network Configuration Protocol (NETCONF) to provide the desired solution of automated and programmable network operations.

YANG model

YANG is a data modeling language used to describe configuration and operational data, remote procedure calls and notifications for network devices. The salient features of YANG are:

- Human-readable format, easy to learn and represent

- Supports definition of operations
- Reusable types and groupings
- Data modularity through modules and submodules
- Supports the definition of operations (RPCs)
- Well-defined versioning rules
- Extensibility through augmentation

For more details of YANG, refer RFC 6020 and 6087.

NETCONF and gRPC (Google Remote Procedute Call) provide a mechanism to exchange configuration and operational data between a client application and a router and the YANG models define a valid structure for the data (that is being exchanged).

Protocol	Transport	Encoding/ Decoding
NETCONF	SSH	XML
gRPC	HTTP/2	XML, JSON

Each feature has a defined YANG model. Cisco-specific YANG models are referred to as synthesized models. Some of the standard bodies, such as IETF , IEEE and Open Config, are working on providing an industry-wide standard YANG models that are referred to as common models.

Components of a YANG Model

A module defines a single data model. However, a module can reference definitions in other modules and submodules by using the **import** statement to import external modules or the **include** statement to include one or more submodules. A module can provide augmentations to another module by using the **augment** statement to define the placement of the new nodes in the data model hierarchy and the **when** statement to define the conditions under which the new nodes are valid. **Prefix** is used when referencing definitions in the imported module.

YANG models are available for configuring a feature and to get operational state (similar to show commands)

This is the configuration YANG model for AAA (denoted by - cfg)

```
(snippet)
module Cisco-IOS-XR-aaa-locald-cfg {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-cfg";


  prefix "aaa-locald-cfg";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  import Cisco-IOS-XR-aaa-lib-cfg { prefix "a1"; }

  /*** META INFORMATION ***/
```

```
  organization "Cisco Systems, Inc.";
          ......................
          ...................... (truncated)
```

This is the operational YANG model for AAA (denoted by -oper)

```
 (snippet)
module Cisco-IOS-XR-aaa-locald-oper {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-oper";


  prefix "aaa-locald-oper";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  include Cisco-IOS-XR-aaa-locald-oper-sub1 {
    revision-date 2015-01-07;
  }

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";
    ......................
    ...................... (truncated)
```

Note A module may include any number of sub-modules, but each sub-module may belong to only one module. The names of all standard modules and sub-modules must be unique.

Structure of YANG models

YANG data models can be represented in a hierarchical, tree-based structure with nodes, which makes them more easily understandable. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes. The nodes types (for data modeling) are:

- leaf node - contains a single value of a specific type
- list node - contains a sequence of list entries, each of which is uniquely identified by one or more key leafs
- leaf-list node - contains a sequence of leaf nodes
- container node - contains a grouping of related nodes containing only child nodes, which can be any of the four node types

Data Types

YANG defines data types for leaf values. These data types help the user in understanding the relevant input for a leaf.

Name	Description
binary	Any binary data
bits	A set of bits or flags
boolean	"true" or "false"
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	Enumerated strings
identityref	A reference to an abstract identity
instance-identifier	References a data tree node
int (integer-defined values)	8-bit, 16-bit, 32-bit, 64-bit signed integers
leafref	A reference to a leaf instance
uint	8-bit, 16-bit, 32-bit, 64-bit unsigned intergers
string	Human-readable string
union	Choice of member types

Supported YANG models

The complete list of the supported IOSXR YANG models are:

`https://github.com/YangModels/yang/tree/master/vendor/cisco/xr`

Introduction to NETCONF

NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. 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 typically running as part of a network manager. The server is typically a network device (router).

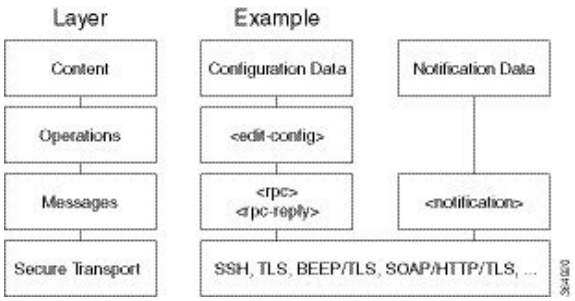
For more NETCONF details, refer RFC 6241.

NETCONF sessions

A NETCONF session is the logical connection between a network administrator or network configuration application and a network device. Global configuration attributes can be changed during any authorized session, and 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 the close or kill messages.

NETCONF Layers

Figure 1: NETCONF Layers



NETCONF can be partitioned into four 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 server

NETCONF Operations

NETCONF defines the existence of one or more configuration datastores and allows configuration operations on them. A configuration datastore is defined as the 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.

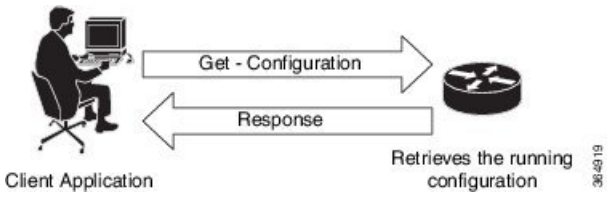
IOS XR NETCONF supports the following operations:

- <get-config>—Retrieves all or part of a specified configuration from a named data store
- <get>—Retrieves running configuration and device state information
- <edit-config>—Loads all or part of a specified configuration to the specified target configuration
- <get-schema>—Retrieves the required schema from the router

NETCONF Operations: Example

This example shows how a NETCONF <get-config> request works.

Figure 2: <get-config> request



The send message request is to get the current configuration of CDP running on the router. The return message includes the current CDP configuration.

NETCONF request (client to server)	NETCONF reply (server to client)
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"> <get-config> <source><running/></source> <filter> <cdp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"/> </filter> </get-config> </rpc>	<?xml version="1.0"?> <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"> <data> <cdp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"> <timer>10</timer> <enable>true</enable> <log-adjacency></log-adjacency> <hold-time>200</hold-time> <advertise-v1-only></advertise-v1-only> </cdp> </data> </rpc-reply>

The RPC element is used to enclose a NETCONF request sent from the client to the server. The <rpc> element has a mandatory attribute *message-id*, which is a string chosen by the sender of the RPC that will commonly encode a monotonically increasing integer. The receiver of the RPC does not decode or interpret this string but simply saves it to be used as a *message-id* attribute in any resulting <rpc-reply> message. The sender MUST ensure that the *message-id* value is normalized. The RPC reply message contains the same message-id when the client receives information from the server.

Subtree Filtering

XML subtree filtering is a mechanism that allows an application to select particular XML subtrees to include in the <rpc-reply> for a <get> or <get-config> operation.

Subtree Filter Components

A subtree filter is comprised of XML elements and their XML attributes. Elements that can be present in a subtree filter are:

- Namespace selection - A namespace is considered to match (for filter purposes) if the XML namespace associated with a particular node within the filter element is the same as in the underlying data model. A namespace selection cannot be used by itself; at least one element must be specified in the filter if any elements are to be included in the filter output.

  ```
  Example:

  <filter type="subtree">
     <top xmlns="http://example.com/schema/1.2/config"/>
  </filter>
  ```

- Attribute match expressions -Filtering is done by matching a specified attribute value. This filtering with the *Match* attribute can be specified only in Table classes.

  ```
   Example:
  ifName is the attribute match expression
  <filter type="subtree">
         <t:top xmlns:t="http://example.com/schema/1.2/config">
           <t:interfaces>
             <t:interface t:ifName="eth0"/>
           </t:interfaces>
         </t:top>
       </filter>
  ```

- Containment Nodes - Filtering is done by specifying nodes (classes) that have child nodes (classes). This filtering is by specifying container classes.

```
 Example: top is a containment node
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users/>
       </top>
     </filter>
```

- Selection Nodes - Filtering is done by specifying leaf nodes. This filtering specifies leaf classes.

```
 Example: users is a selection node (in the containment node - top)
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users/>
       </top>
     </filter>
```

- Content Match Nodes - Filtering is done by exactly matching the content of a leaf node. This filtering is done by specifying naming the class value for table classes.

```
Example: name is the content match node (in the containment node - top and the selection
 node - user)
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users>
           <user>
             <name>fred</name>
           </user>
         </users>
       </top>
     </filter>
```

gRPC

gRPC is a language-neutral, open source, RPC (Remote Procedute Call) system developed by Google. By default, it uses protocol buffers as the binary serialization protocol. It can be used with other serialization protocols as well such as JSON, XML etc. The user needs to define the structure by defining protocol buffer message types in*.proto* files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs.

gRPC encodes requests and responses in binary. Although Protobufs was the only format supported in the initial release, gRPC is extensible to other content types. The Protobuf binary data object in gRPC is transported using HTTP/2 (RFC 7540). HTTP/2 is a replacement for HTTP that has been optimized for high performance. HTTP/2 provides many powerful capabilities including bidirectional streaming, flow control, header compression and multi-plexing. gRPC builds on those features, adding libraries for application-layer flow-control, load-balancing and call-cancellation.

gRPC supports distributed applications and services between a client and server. gRPC provides the infrastructure to build a device management service to exchange configuration and operational data between a client and a server in which the structure of the data is defined by YANG models.

Cisco gRPC IDL

The protocol buffers interface definition language (IDL) is used to define service methods, and define parameters and return types as protocol buffer message types.

gRPC requests can be encoded and sent across to the router using JSON. gRPC IDL also supports the exchange of CLI.

For gRPC transport, gRPC IDL is defined in .proto format. Clients can invoke the RPC calls defined in the IDL to program XR. The supported operations are - Get, Merge, Delete, Replace. The gRPC JSON arguments are defined in the IDL.

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

    rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};

    rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};

    rpc DeleteConfig(ConfigArgs) returns(ConfigReply) {};

    rpc ReplaceConfig(ConfigArgs) returns(ConfigReply) {};

    rpc CliConfig(CliConfigArgs) returns(CliConfigReply) {};

}
```

gRPC Operations

- oper get-config—Retrieves a configuration
- oper merge-config— Appends to an existing configuration
- oper delete-config—Deletes a configuration
- oper replace-config—Modifies a part of an existing configuration
- oper get-oper—Gets operational data using JSON
- oper cli-config—Performs a configuration
- oper showcmdtextoutput