Cisco IOS XR Programmability Configuration Guide for ASR 9000 Series Router

First Published: 2016-08-31

Last Modified: 2016-11-14

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see What's New in Cisco Product Documentation.

To receive new and revised Cisco technical content directly to your desktop, you can subscribe to the What's New in Cisco Product Documentation RSS feed. RSS feeds are a free service.

CONTENTS

CHAPTER 1

Programmatic Configuration Using Data Models

Data models are a programmatic way of configuring and collecting operational data of a network device. They replace the process of manual configuration, which is proprietary, and highly text-based.

Data Models—Scope, Need, and Benefits

Scope

Data models can be used to automate configuration tasks across heterogeneous devices in a network.

Data models handle the following types of data on routers (RFC 6244):

- **Configuration data:** A set of writable data that is required to transform a system from an initial default state into its current state. For example, configuring entries of the IP routing tables, configuring the interface MTU to use a specific value, configuring an ethernet interface to run at a given speed, and so on.
- **Operational state data:** A set of data that is obtained by the system at runtime and influences the behavior of the system in a manner similar to configuration data. However, in contrast to configuration data, operational state data is transient. The data is modified by interactions with internal components or other systems using specialized protocols. For example, entries obtained from routing protocols such as OSPF, attributes of the network interfaces, and so on.

Data models provide a well-defined hierarchy of the configurational and operational data of a router. The data models are programmed to provide a common framework of configurations to be deployed across networks. This common framework helps to program and manage a network with ease.

For more information about Data Models, see RFC 6244.

Need

Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly.

CLIs are widely used for configuring and extracting the operational details of a router. But the general mechanism of CLI scraping is not flexible and optimal. A small change in the configuration requires re-writing scripts numerous times. Bulk configuration changes through CLIs are cumbersome and error-prone. These limitations restrict automation and scale.

To overcome these limitations, Cisco IOS XR supports a programmatic way of writing configurations to any network device using data models.

Data models help to manipulate configuration data and retrieve operational data. The data models replace the process of manual configuration and are written in an industry-defined language. Although configurations using CLIs are easier and human-readable, automating the configuration using data models results in scalability. To get these data models, see Obtain Data Models, on page 5

The data models provides access to the capabilities of the devices in a network using Network Configuration Protocol (NETCONF) or gRPC (google-defined Remote Procedure Calls) protocols. The operations on the router are carried out by the protocols using YANG models to automate and programme operations in a network. To enable the protocol, see Enable Protocol, on page 6.

The process of automating configurations in a network is accomplished using the core components - router, client application, YANG model and communication protocols. For more information about the core components, see Components to Use Data Models, on page 13.

Benefits

Configuring routers using data models overcomes drawbacks posed by traditional router management because the data models:

- Provide a common model for configuration and operational state data.
- Use protocols to communicate with the routers to get, manipulate and delete configurations in a network.
- Automate configuration and operation of multiple routers across the network.

Process for using Data Models

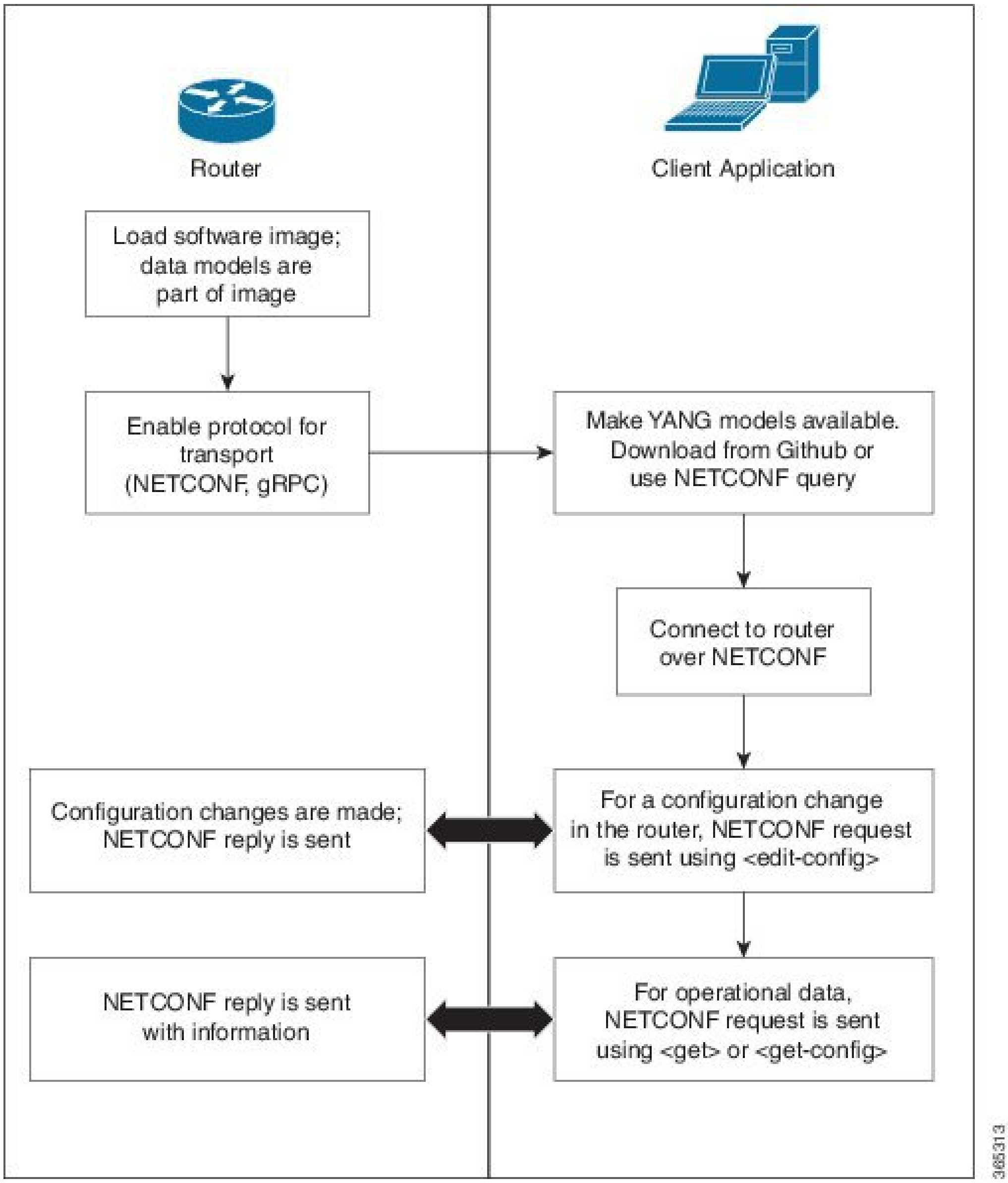
The process for using data models involves:

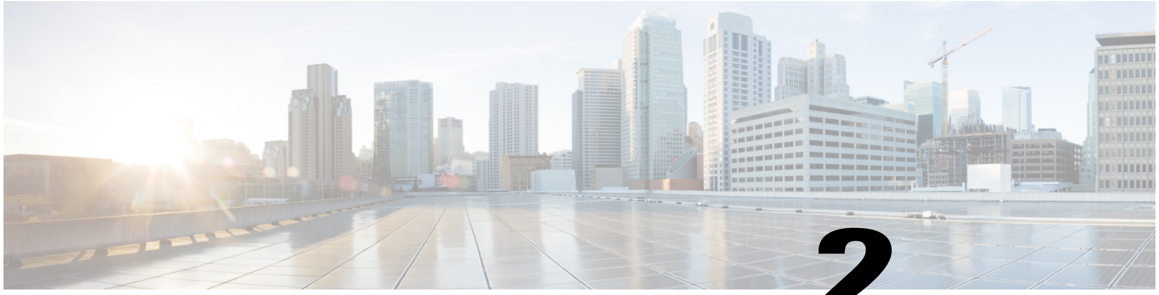
- Obtain the data models.
- Establish a connection between the router and the client using communication protocols such as NETCONF or gRPC.
- Manage the configuration of the router from the client using data models.

Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization before setting up any configuration.

Figure 1 shows the tasks involved in using data models.

Figure 1: Process for Using Data Models


Router
Client Application
Load software image; data models are part of image
Enable protocol for transport (NETCONF, gRPC)
Make YANG models available. Download from Github or use NETCONF query
Connect to router over NETCONF
Configuration changes are made; NETCONF reply is sent
For a configuration change in the router, NETCONF request is sent using <edit-config>
NETCONF reply is sent with information
For operational data, NETCONF request is sent using <get> or <get-config>
365313



CHAPTER 2

Using Data Models

Using data models involves three tasks:

Obtain Data Models

The data models are available in the mgbl pie software package. Installing a package on the router installs specific features that are part of that package. Cisco IOS XR software is divided into various software packages to select the features to run on the router. Each package contains components that perform a specific set of router functions, such as routing, security, and so on.

1 Load the mgbl pie software image on the router.

2 Verify that the data models are available using `netconf-monitoring` request.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <get>
   <filter type="subtree">
     <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
     <schemas/>
     </netconf-state>
   </filter>
 </get>
</rpc>
```

All XR YANG models are displayed.

Note Calvados models, which are the System Admin models, are copied from the Calvados environment to the XR environment at the `/pkg/yang/` location only after the protocol (NETCONF, gRPC) is enabled.

The YANG models can be retrieved from the router without logging into the router using **get-schema** command:

Get Schema List (data will be used in step 2).

```
<get>
<filter type="subtree">
```

```
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas/>
</netconf-state>
</filter>
</get>
</rpc>
```

All the models on the router is displayed.

```
TRACE: 2016/06/13 11:11:42 transport.go:104: Reading from connection
TRACE: 2016/06/13 11:11:42 gnc_main.go:587: Session established (Id: 1009461378)
TRACE: 2016/06/13 11:11:42 session.go:93: Request:
<rpc message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get><filter type="subtree"><netconf-state

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring"><schemas/></netconf-state></filter></get></rpc>
TRACE: 2016/06/13 11:11:42 transport.go:104: Reading from connection
TRACE: 2016/06/13 11:11:42 session.go:117:
Response:
#143589
<rpc-reply message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas>
<schema>
<identifier>Cisco-IOS-XR-crypto-sam-oper</identifier>
<version>2015-01-07</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
<location>NETCONF</location>
</schema>
<schema>
<identifier>Cisco-IOS-XR-crypto-sam-oper-sub1</identifier>
<version>2015-01-07</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
<location>NETCONF</location>
</schema>
<schema>
<identifier>Cisco-IOS-XR-snmp-agent-oper</identifier>
<version>2015-10-08</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
<location>NETCONF</location>
</schema>
------------<truncated>--------------
```

For more information about structure of data models, see YANG Module, on page 13.

What To Do Next:

Enable the protocol to establish connection between the router and the client application.

Enable Protocol

The router communicates with the client application using protocols. On the router and client application, enable a communication protocol based on the requirement:

- NETCONF
- gRPC

For more information about protocols, see Communication Protocols, on page 17.

Enable NETCONF over SSH Protocol

NETCONF is an XML-based protocol used over Secure Shell (SSH) transport to configure a network. The client applications use this protocol to request information from the router, and make configuration changes to the router.

For more information about NETCONF, see NETCONF Protocol, on page 17.

Pre-requisites:

- Software package k9sec pie is installed on the router.
- Software package mgbl pie is installed on the router.
- Crypto keys are generated.

To enable the NETCONF protocol, complete these steps:

1. Enable NETCONF protocol over an SSH connection.

   ```
   ssh server v2
   ssh server netconf
   netconf agent tty
   netconf-yang agent ssh
   ```

   The default port number of 830 is used. A different port within the range of 1 to 65535 can be specified if required.

2. Set the session parameters.

   ```
   router (config)# netconf-yang agent session {  limit value |  absolute-timeout value |
    idle-timeout value }
   ```

   where:

   - **limit value:** sets the maximum count for concurrent netconf-yang sessions. The range is from 1 to 1024.
   - **absolute-timeout value:** sets the absolute session lifetime, in minutes. The range is from 1 to 1440.
   - **idle-timeout value:** sets the idle session lifetime, in minutes. The range is from 1 to 1440.

3. Verify configuration settings for statistics and clients.

   ```
   router (config)# show netconf-yang statistics
   router (config)# show netconf-yang clients
   ```

Example: Enable NETCONF

```
config
 netconf-yang agent ssh
  ssh server netconf port 830
!
```

Example: Verify Configuration Using Statistics

After the NETCONF request is sent, use **show netconf-yang statistics** command to verify the configuration.

```
show netconf-yang statistics
Summary statistics        requests|            total time|   min time per request|   max
time per request|   avg time per request|
other                             0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
close-session                     4|       0h  0m  0s   3ms|       0h  0m  0s   0ms|
```

```
 0h  0m  0s   1ms|       0h  0m  0s   0ms|
kill-session                      0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
get-schema                        0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
get                               0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s
get-config                        1|       0h  0m  0s   1ms|       0h  0m  0s   1ms|
 0h  0m  0s   1ms|       0h  0m  0s   1ms|
edit-config                       3|       0h  0m  0s   2ms|       0h  0m  0s   0ms|
 0h  0m  0s   1ms|       0h  0m  0s   0ms|
commit                            0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
cancel-commit                     0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
lock                              0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
unlock                            0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
discard-changes                   0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
validate                          0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|
```

Example: Verify Configuration Using Clients

```
show netconf-yang clients
client session ID|   NC version|    client connect time|        last OP time|        last
OP type|    <lock>|
22969|                 1.1|         0d  0h  0m  2s|             11:11:24|
close-session|        No|
15389|
```

What To Do Next:

After NETCONF is enabled, use the YANG data models to manage the relevant configurations.

Enable gRPC over HTTP/2 Protocol

Google-defined remote procedure call (gRPC) is an open-source RPC framework. gRPC supports IPv4 and v6 address families.

For more information about gRPC, see gRPC Protocol, on page 19.

Pre-requisite:

- Configure TLS.

Note It is recommended to configure TLS. Enabling gRPC protocol uses the default HTTP/2 transport with no TLS enabled on TCP. gRPC mandates AAA authentication and authorization for all gRPC requests. If TLS is not configured, the authentication credentials are transferred over the network unencrypted. Enabling TLS ensures that the credentials are secure and encrypted. Non-TLS mode can only be used in secure internal network.

- Software package mgbl pie is installed on the router.

To enable the gRPC protocol, complete these steps:

1 Enable gRPC over an HTTP/2 connection.

```
router# configure
router (config)# grpc
```

2 Enable access to a specified port number.

```
router (config)# port <port-number>
```

The <port-number> range is from 57344 to 57999. If a port number is unavailable, an error is displayed.

3 Set the session parameters.

```
router (config)# grpc{ address-family | max-request-per-user | max-request-total | tls}
```

where:

- **address-family:** sets the address family identifier type.
- **max-request-per-user:** sets the maximum concurrent requests for each user.
- **max-request-total:** sets the maximum possible total number of concurrent requests.
- **tls:** enables Transport Layer Security (TLS).

What To Do Next:

After gRPC is enabled, use the YANG data models to manage the relevant configurations.

Manage Configurations using Data Model

From the client application, use data models to manage the configurations of the router.

Prerequisites

- Software packages k9sec pie and mgbl are installed on the router.
- NETCONF or gRPC protocols are enabled on the client and the router.

To manage configurations using data models, complete these steps:

1 Use a YANG tool to import the data model on the client application.

2 Configure the router by modifying the values of the data model using the YANG tool.

For more information on the values of the data models that can be configured, see Structure of YANG Models, on page 15.

Example: Configure CDP

In this example, you use the data model for CDP and configure CDP with the values as shown in the table:

CDP parameter	Description	Desired value for parameter
CDP Version	Specifies the version used to communicate with the neighboring devices	v1
Hold time	Specifies the duration for which the receiving device to hold the CDP packet	200 ms

CDP parameter	Description	Desired value for parameter
Timer	Specifies how often the software sends CDP updates	80 ms
Log Adjacency Table	Logs changes in the adjacency table. When CDP adjacency table logging is enabled, a syslog is generated each time a CDP neighbor is added or removed	enable

1 Download the configuration YANG data model for CDP `Cisco-IOS-XR-cdp-cfg.yang` from the router. To download the data model, see Obtain Data Models, on page 5.

2 Import the data model to the client application using any YANG tool.

3 Modify the leaf nodes of the data model:

- enable (to enable cdp)
- holdtime
- timer
- advertise v1 only
- log adjacency

Configure CDP Using NETCONF

In this example, you use the data model for CDP and configure CDP using NETCONF RPC request:

```
<edit-config>
  <target>
   <candidate/>
  </target>
  <config xmlns:xc="urn:ietf:params:xml:n:netconf:base:1.0">
   <cdp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg">
   <timer>80</timer>
   <enable>true</enable>
   <log-adjacency></log-adjacency>
   <hold-time>200</holdtime>
   <advertise-v1-only></advertise-v1-only>
  </cdp>
 </config>
</edit-config>
```

Note CDP can also be configured under the interface configuration by augmenting the interface manager. Use the `Cisco-IOS-XR-ifmgr-cfg` YANG model to configure CDP under the interface configuration.

Note Flexible CLI group and apply-group configuration can be created using NETCONF YANG client. The flexible CLI configuration groups provide the ability to minimize repetitive configurations by defining a series of configuration statements in a configuration group, and then applying this group to multiple hierarchical levels in the router configuration tree. For example, see a use case to Configure a Group for Bundle Interfaces using Flexible CLI, on page 26.

Configure CDP Using gRPC

In this example, you use the data model for CDP and configure CDP using gRPC MergeConfig RPC request:

```
{
 "Cisco-IOS-XR-cdp-cfg:cdp": {
  "timer": 50,
  "enable": true,
  "log-adjacency": [
   null
  ],
  "hold-time": 180,
  "advertise-v1-only": [
   null
  ]
 }
}
```

Note CDP can also be configured under the interface configuration by augmenting the interface manager. Use the `Cisco-IOS-XR-ifmgr-cfg` YANG model to configure CDP under the interface configuration.

CHAPTER 3

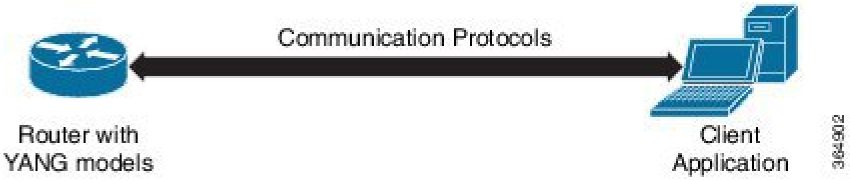
Components to Use Data Models

The process of automating configurations in a network involves the use of these core components:

- **Client application:** manages and monitors the configuration of the devices in the network.
- **Router:** acts as a server, responds to requests from the client application and configures the devices in the network.
- **YANG module:** describes configuration and operational data of the router.
- **Communication protocol:** provides mechanisms to install, manipulate, and delete the configuration of network devices.

Figure 2 shows the interplay of the core components.

Figure 2: Components in Using Data Models



This chapter describes these two components:

YANG Module

A YANG module 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, 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/xr/.
- **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. Cisco-supported OC models are:
    - OC-BGP
    - OC-Route Policy
    - OC-MPLS
    - OC-Interfaces
    - OC-IF-Ethernet
    - OC-IF-Aggregate

For a list of supported OC models and their representation, see https://github.com/openconfig/public/tree/master/release/models.

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

Components of a YANG Module

A YANG module 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 and retrieve the operational state of the router.

Note The gRPC YANG path or JSON data is based on YANG module name and not YANG namespace.

Example: Configuration YANG Model for AAA

The YANG models used to configure a feature is 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)
```

Example: Operational YANG Model for AAA

The YANG models used to retrieve operational data is 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 can include any number of sub-modules; each sub-module belongs 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. 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

Example: Structure of CDP Data Model

Cisco Discovery Protocol (CDP) configuration has an inherent augmented model (interface-configuration). The augmentation indicates that CDP can be configured at both the global configuration level and the interface configuration level. The data model for CDP interface manager in tree structure is:

```
module: Cisco-IOS-XR-ifmgr-cfg
   +--rw global-interface-configuration
   |  +--rw link-status?   Link-status-enum
   +--rw interface-configurations
      +--rw interface-configuration* [active interface-name]
         +--rw dampening
         |  +--rw args?                 enumeration
         |  +--rw half-life?            uint32
         |  +--rw reuse-threshold?      uint32
         |  +--rw suppress-threshold?   uint32
         |  +--rw suppress-time?        uint32
         |  +--rw restart-penalty?      uint32
         +--rw mtus
         |  +--rw mtu* [owner]
         |     +--rw owner    xr:Cisco-ios-xr-string
         |     +--rw mtu      uint32
         +--rw encapsulation
         |  +--rw encapsulation?         string
         |  +--rw capsulation-options?   int32
         +--rw shutdown?                      empty
         +--rw interface-virtual?             empty
         +--rw secondary-admin-state?         Secondary-admin-state-enum
         +--rw interface-mode-non-physical?   Interface-mode-enum
         +--rw bandwidth?                     int32
         +--rw link-status?                   empty
         +--rw description?                   string
         +--rw active                         Interface-active
         +--rw interface-name                 xr:Interface-name
```

CDP Operational YANG:

```
module: Cisco-IOS-XR-cdp-oper
   +--ro cdp
      +--ro nodes
         +--ro node* [node-name]
            +--ro neighbors
            |  +--ro details
            |  |  +--ro detail*
            |  |     +--ro interface-name?   xr:Interface-name
            |  |     +--ro device-id?        string
            |  |     +--ro cdp-neighbor*
            |  |        +--ro detail
            |  |        |  +--ro network-addresses
            |  |        |  |  +--ro cdp-addr-entry*
            |  |        |  |     +--ro address
            |  |        |  |        +--ro address-type?   Cdp-l3-addr-protocol
            |  |        |  |        +--ro ipv4-address?   inet:ipv4-address
            |  |        |  |        +--ro ipv6-address?   In6-addr
            |  |        |  +--ro protocol-hello-list
            |  |        |  |  +--ro cdp-prot-hello-entry*
            |  |        |  |     +--ro hello-message?   yang:hex-string
            |  |        |  +--ro version?                string
            |  |        |  +--ro vtp-domain?             string
            |  |        |  +--ro native-vlan?            uint32
            |  |        |  +--ro duplex?                 Cdp-duplex
            |  |        |  +--ro system-name?            string
            |  |        +--ro receiving-interface-name?   xr:Interface-name
            |  |        +--ro device-id?                  string
            |  |        +--ro port-id?                    string
            |  |        +--ro header-version?             uint8
            |  |        +--ro hold-time?                  uint16
            |  |        +--ro capabilities?               string
            |  |        +--ro platform?                   string

            ..............................................  (truncated)
```

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 one of these protocols :

- Network Configuration Protocol (NETCONF)
- gRPC (google-defined Remote Procedure Calls)

The transport and encoding mechanisms for these two protocols are shown in the table:

Protocol	Transport	Encoding/ Decoding
NETCONF	ssh	xml
gRPC	http/2	json

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. 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.

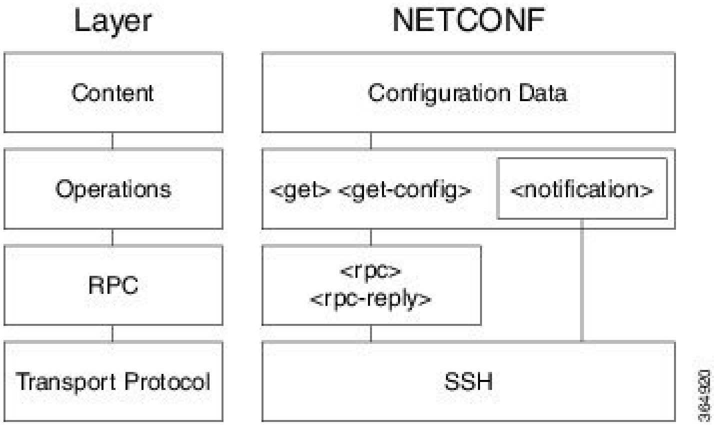
NETCONF Session

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 3: 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

For more information about NETCONF, refer RFC 6241.

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.

NETCONF Operation	Description
<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

NETCONF Operation	Description
<get-schema>	Retrieves the required schema from the router

Example: NETCONF Operation to Get Configuration

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

The client initiates a message to get the current configuration of CDP running on the router. The router responds with the current CDP configuration.

Netconf Request (Client to Router)	Netconf Response (Router 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>`
	`#22`
	` </data>`
	`</rpc-reply>`

The <rpc> element in the request and response messages enclose a NETCONF request sent between the client and the router. The `message-id` attribute in the <rpc> element is mandatory. This attribute is a string chosen by the sender and encodes an integer. The receiver of the <rpc> element does not decode or interpret this string but simply saves it to be used in the <rpc-reply> message. The sender must ensure that the `message-id` value is normalized. When the client receives information from the server, the <rpc-reply> message contains the same `message-id`.

gRPC Protocol

gRPC is an open-source RPC framework. It is based on Protocol Buffers (Protobuf), which is an open source binary serialization protocol. gRPC provides a flexible, efficient, automated mechanism for serializing structured data, like XML, but is smaller and simpler to use. 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. gRPC is extensible to other content types along with Protobuf. The Protobuf binary data object in gRPC is transported over HTTP/2.

Note It is recommended to configure TLS before enabling gRPC. Enabling gRPC protocol uses the default HTTP/2 transport with no TLS enabled on TCP. gRPC mandates AAA authentication and authorization for all gRPC requests. If TLS is not configured, the authentication credentials are transferred over the network unencrypted. Non-TLS mode can only be used in secure internal network.

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. The structure of the data is defined by YANG models.

Cisco gRPC IDL uses the protocol buffers interface definition language (IDL) to define service methods, and define parameters and return types as protocol buffer message types. The gRPC requests are encoded and sent to the router using JSON. Clients can invoke the RPC calls defined in the IDL to program the router.

The following example shows the syntax of the proto file for a gRPC configuration:

```
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) {};

    rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

}


service gRPCExec {
    rpc ShowCmdTextOutput(ShowCmdArgs) returns(stream ShowCmdTextReply) {};
}

message ConfigGetArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

message ConfigGetReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}

message GetOperArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

message GetOperReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}


message ConfigArgs {
    int64 ReqId = 1;
    string yangjson = 2;
}

message ConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CliConfigArgs {
    int64 ReqId = 1;
    string cli = 2;
}

message CliConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}


message ShowCmdArgs {
     int64 ReqId = 1;
     string cli = 2;
}

message ShowCmdTextReply {
    int64 ResReqId =1;
    string output = 2;
    string errors = 3;
}
```

gRPC Operations

The gRPC operations include:

gRPC Operation	Description
GetConfig	Retrieves a configuration
MergeConfig	Appends to an existing configuration
DeleteConfig	Deletes a configuration
ReplaceConfig	Modifies a part of an existing configuration
GetOper	Gets operational data using JSON
CliConfig	Invokes the CLI configuration
ShowCmdTextOutput	Displays the output of show command

Example: Get Configuration for a Specific Interface

This example shows getting configuration for a specific interface using gRPC GetConfig operation.

```
{
    "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
        "interface-configuration": [
            {
                "active": "act",
                "interface-name": "HundredGigE0/3/0/0"
            }
        ]
    }
}
```

Example: Delete Configuration for CDP Container

This example shows how a gRPC DeleteConfig operation deletes a CDP container and a leaf within the container. The DeleteConfig argument identifies the resource using the YANG node. The value of the YANG node is ignored and set to null.

In this example, a CDP container is deleted:

```
{
"Cisco-IOS-XR-cdp-cfg:cdp": [null]
}
```

In this example, a leaf value for `hold-time` in the CDP container is deleted:

```
{
"Cisco-IOS-XR-cdp-cfg:cdp":
{
"hold-time": [null]
}
}
```

Example: Merge Configuration for CDP Timer

This example shows merging configuration for CDP timer using gRPC MergeConfig operation.

```
{
    "Cisco-IOS-XR-cdp-cfg:cdp": {
        "timer": 50
    }
}
```

Example: Get Operational Data for Interfaces

This example getting operational data for interfaces using gRPC GetOper operation.

```
{
    "Cisco-IOS-XR-ifmgr-oper:interface-properties": [null]
}
```

CHAPTER 4

Use Cases with Data Models

In this section, certain uses cases with data models are described.

Delete BGP Neighbor

In this use case, you delete a BGP neighbor using YANG models.

1 Using standard YANG tools, get the configuration using the NETCONF <get-config> operation in YANG format.

```
<get-config>
    <source>
      <running/>
    </source>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <bgp xmlns="http://openconfig.net/yang/bgp">
 </get-config>

router bgp 1000
 bgp confederation peers
  65002
 !
 bgp confederation identifier 102
 bgp router-id 1.1.1.1
 bgp graceful-restart restart-time 30
 bgp graceful-restart stalepath-time 30
 bgp graceful-restart
 address-family ipv4 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv4 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
!router bgp 1000
 bgp confederation peers
  65002
 !
 bgp confederation identifier 102
 bgp router-id 1.1.1.1
 bgp graceful-restart restart-time 30
 bgp graceful-restart stalepath-time 30
 bgp graceful-restart
 address-family ipv4 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv4 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
!
```

2. Change the configuration <edit-config> operation.

```
<edit-config>
    <target>
      <candidate/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <bgp xmlns="http://openconfig.net/yang/bgp">
        <global>
          <config>
            <as xc:operation="delete">1000</as>
          </config>
        </global>
      </bgp>
    </config>
  </edit-config>
```

3. Send the <edit-config> request through NETCONF SSH to the router.

4. Verify that the configuration changes is successful and the BGP neighbor is deleted.

Note BGP configuration can be fetched using gRPC GetConfig operation:

```
{
  “bgp:bgp”: [
 null
  ]
}
```

Delete BGP configuration use gRPC DeleteConfig:

```
{
    "bgp:bgp": {
        "global": {
            "config": {
                "as": [
                    null
                ]
            }
        }
    }
}
```

Request for AAA Access Details

In this use case, you use a Calvados model to view AAA access details.

Prerequisites

- Ensure that the user is added to the Calvados environment. This is because even if the user is added to the XR environment and has `root-lr` permissions, access to Calvados models is denied.
- Establish a NETCONF or gRPC connection between the router and the client application.

Note The gRPC YANG path or JSON data is based on YANG module name and not YANG namespace.

1 Using standard YANG tools, send a request to the router from the client using the NETCONF <get> operation.

```
[ Request ]
<get>
  <filter type="subtree">
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
      <privileged-access xmlns="http://www.cisco.com/calvados/aaa_show"/>
    </aaa>
  </filter>
</get>
```

2 Verify the response sent by the router to the client.

```
[ Response ]
<?xml version="1.0" encoding="UTF-8"?><data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
 xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
   <privileged-access xmlns="http://www.cisco.com/calvados/aaa_show">
    <shell-access>None</shell-access>
    <first-user>root</first-user>
    <first-user-change>No</first-user-change>
    <current-disaster-recovery-user>root</current-disaster-recovery-user>
   </privileged-access>
  </aaa>
 </data>
```

Note To accomplish this task using gRPC GetOper request:

```
{
    "tailf-aaa:aaa": {
        "aaa_show:privileged-access": [
            null
        ]
    }
}
```

gRPC GetOper response:

```
{
 "tailf-aaa:aaa": {
  "aaa_show:privileged-access": {
   "shell-access": "None",
   "first-user": "root",
   "first-user-change": "No",
   "current-disaster-recovery-user": "root"
  }
 }
}
```

Configure a Group for Bundle Interfaces using Flexible CLI

In this use case, you use the flexible CLI functionality to define a group with the associated configuration commands for bundle interfaces. This configuration can be applied to the bundle interfaces based on requirement. The group configuration eliminates the need to create configuration for each bundle interface.

The configuration groups utilize regular expressions that are checked for a match at multiple submodes of the configuration tree based on where the group is applied within the hierarchy. If a match is found at a configuration submode, the corresponding configuration defined in the group is inherited within the matched submode. Flexible CLI configuration groups also provide an auto-inheritance feature. This feature implies that any change done to a CLI configuration group is automatically applied to the configuration in any matched submodes that have an apply-group at that hierarchical level. This allows you to make a configuration change or addition once, and have it applied automatically in multiple locations, depending on where you have applied the flexible CLI configuration group.

1 Create the group for bundle interfaces:

a Create group using NETCONF client.

```
<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">
<groups xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-group-cfg">
<group>
<group-name>GRP_BUNDLE_CFG</group-name>
 <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg" >
  <interface-configuration>
  <active>act</active>
  <interface-name>Bun.*</interface-name>
   <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
     <addresses>
      <primary>
```

```
       <address>192.168.0.100</address>
       <netmask>255.255.255.0</netmask>
      </primary>
     </addresses>
   </ipv4-network>
   <bundle xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-bundlemgr-cfg">
     <wait-while>1600</wait-while>
     <maximum-active>
      <links>
       <links>50</links>
       <max-active-links-mode>default</max-active-links-mode>
      </links>
     </maximum-active>
     <minimum-active>
      <links>1</links>
     </minimum-active>
   </bundle>
  </interface-configuration>
 </interface-configurations>
</group>
</groups>
</config>
</edit-config>
</rpc>
##
```

b Commit the configuration.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
##
```

2 Apply the group to bundle interfaces `Bundle-Ether1` and `Bundle-Ether2`.

a

```
<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">
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
    <active>act</active>
    <interface-name>Bundle-Ether1</interface-name>
    <apply-groups>
        <apply-group>GRP_BUNDLE_CFG</apply-group>
    </apply-groups>
   </interface-configuration>
   <interface-configuration>
    <active>act</active>
    <interface-name>Bundle-Ether2</interface-name>
    <apply-groups>
        <apply-group>GRP_BUNDLE_CFG</apply-group>
    </apply-groups>
   </interface-configuration>
</interface-configurations>
</config>
</edit-config>
</rpc>
##
```

b Commit the configuration.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
##
```

3 Verify the applied configuration using **get-config** request from NETCONF client.

The NETCONF protocol supports concise view and inheritance view. The concise view is the default view and returns the configuration with flexible CLI groups as present. The inheritance view returns the configuration with expanded configuration for the corresponding flexible CLI groups.

- To retrieve the concise configuration, use **get-config** with source `<running/>` request:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
</interface-configurations>
</filter>
</get-config>
</rpc>
##
```

- To retrieve the inherited configuration, use **get-config** with source `<running-inheritance/>` request:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running-inheritance/>
</source>
<filter type="subtree">
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
</interface-configurations>
</filter>
</get-config>
</rpc>
##
```