Programmability Configuration Guide for Cisco CRS Routers, IOS XR Release 6.4.x

First Published: 2018-03-30

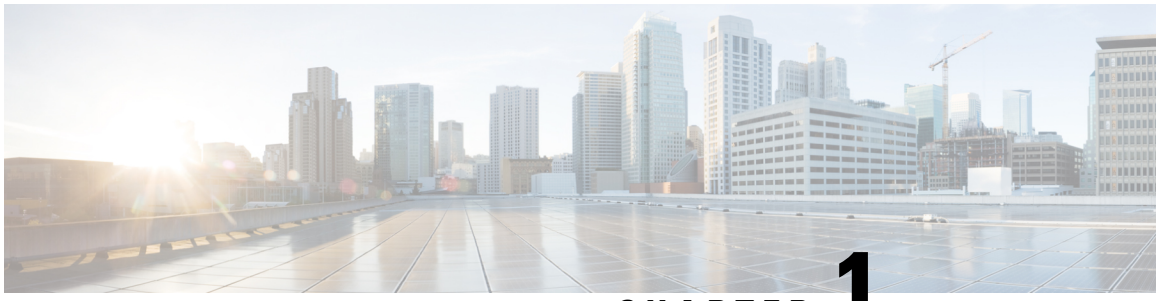
Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

CONTENTS



CHAPTER 1

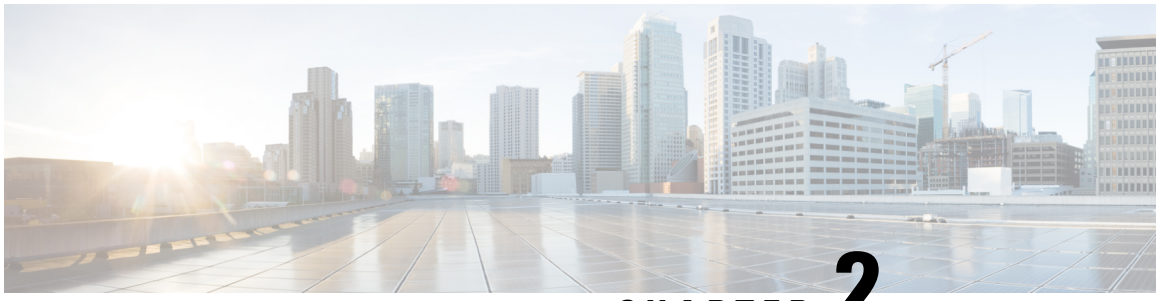
New and Changed Feature Information

This section lists all the new and changed features for the Programmability Configuration Guide.

New and Changed Programmability Features

Note *This software release has reached end-of-life status. For more information, see the* End-of-Life and End-of-Sale Notices.

Feature	Description	Changed in Release	Where Documented
Support for Open Config model - OC-NI Local Routing	This Cisco-supported Open Config models have YANG models defined for configuration and operational data.	Release 6.4.1	*Use Case with Data Models* chapter Use OC Model to Configure Static Route, on page 26
Support for Open Config models - OC-Network-instance, OC-ISIS	These Cisco-supported Open Config models define YANG models for configuration and operational data.	Release 6.4.1	*Components to Use Data Models* chapter YANG Module, on page 13



CHAPTER 2

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 requirements 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.
- **Actions:** A set of NETCONF actions that support robust network-wide configuration transactions. When a change is attempted that affects multiple devices, the NETCONF actions simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.

Data models provide a well-defined hierarchy of the configurational and operational data of a router, and NETCONF actions. 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. Small changes in the configuration require rewriting scripts multiple 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, retrieve operational data, and perform actions. 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 7

The data models provides access to the capabilities of the devices in a network using Network Configuration Protocol (NETCONF). 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 8

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, and perform NETCONF actions.
- 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:

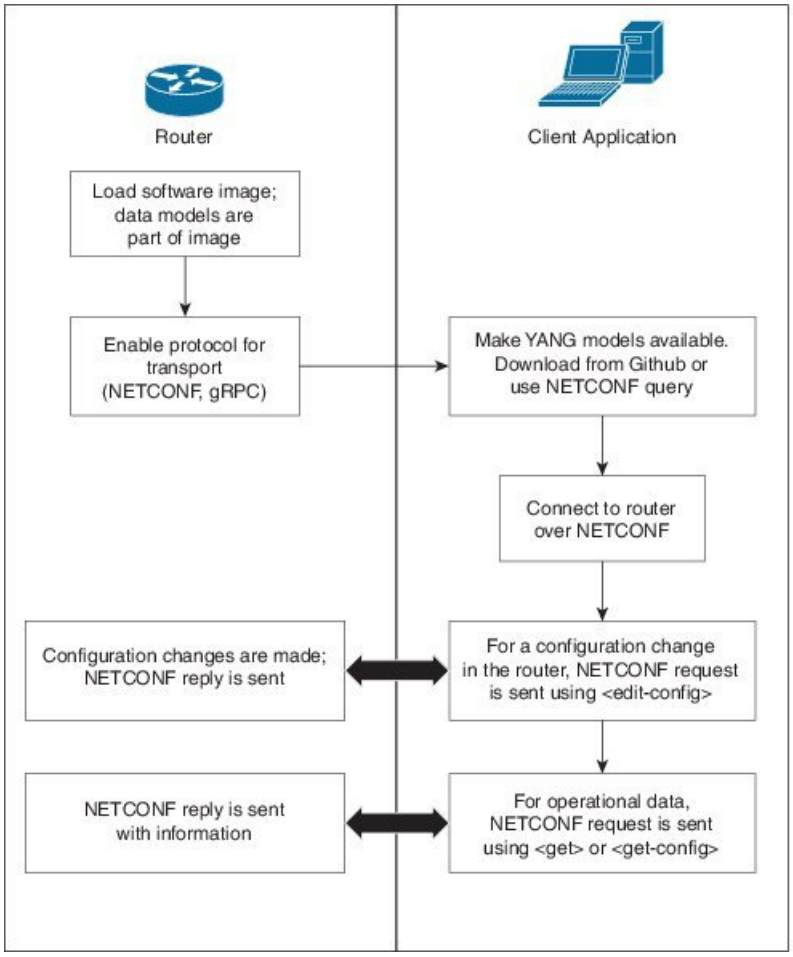
- Obtain the data models.
- Establish a connection between the router and the client using communication protocols such as NETCONF.
- Manage the configuration of the router from the client using data models.

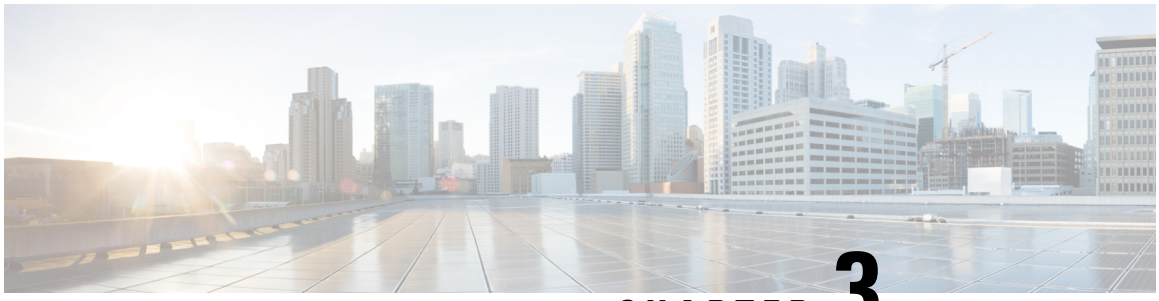
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization using **aaa authorization exec** command before setting up any configuration. For more information about configuring AAA authorization, see the *System Security Configuration Guide for Cisco CRS Routers*.

The following image shows the tasks involved in using data models.

Note gRPC is supported only on 64-bit platforms.

Figure 1: Process for Using Data Models





CHAPTER 3

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.

Pre-requisites:

Ensure that the mgbl pie software image is loaded in the router.

1. 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 IOS XR YANG models are displayed.

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 are 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 the NETCONF communication protocol.

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 18.

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

Enable NETCONF

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

Verify Configuration Using Statistics

After the NETCONF request is sent, use **do 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|
```

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|
```

What To Do Next:

After NETCONF 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 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 16.

Note The OC interface maps all IP configurations for parent interface under a VLAN with index 0. This restricts configuring a sub interface with tag 0.

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
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 7.
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

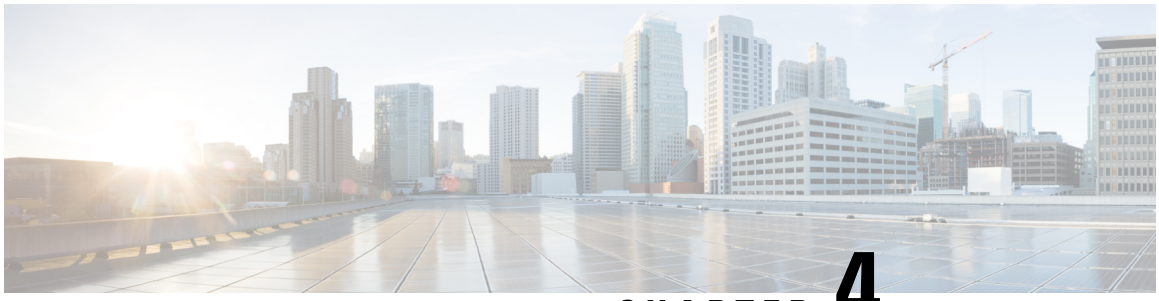
Configure CDP Using NETCONF

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.



CHAPTER 4

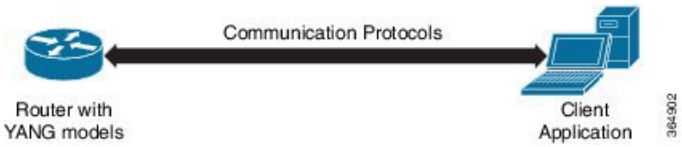
Components to Use Data Models

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, and perform actions.
- **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, on page 13
- Communication Protocols, on page 17
- YANG Actions, on page 22

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, 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 applications.

YANG models can be:

- **Cisco-specific models:** For a list of supported models and their representation, see Github.
- **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 Github.

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, retrieve the operational state of the router, and perform actions.

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.

Example: NETCONF Action for OSPFv3

The YANG models used to perform actions is denoted by -act.

```
 (snippet)
clear ospfv3 1 vrf vrf1 statistics neighbor 2.2.2.2
RPC message based on the new ospfv3 yang model-
 <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <act-ospfv3-instance-vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv6-ospfv3-act">

     <instance>
       <instance-identifier>1</instance-identifier>
        <vrf>
         <vrf-name>vrf1</vrf-name>
         <stats>
          <neighbor>
           <neighbor-id>2.2.2.2</neighbor-id>
          </neighbor>
         </stats>
        </vrf>
     </instance>
   </act-ospfv3-instance-vrf>
 </rpc>
```

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-cdp-cfg
   +--rw cdp
      +--rw timer?               uint32
      +--rw advertise-v1-only?   empty
      +--rw enable?              boolean
      +--rw hold-time?           uint32
      +--rw log-adjacency?       empty
augment /a1:interface-configurations/a1:interface-configuration:
   +--rw cdp
      +--rw enable?   empty
```

In the CDP YANG model, the augmentation is expressed as:

```
augment "/a1:interface-configurations/a1:interface-configuration" {
    container cdp {
      description "Interface specific CDP configuration";
      leaf enable {
        type empty;
        description "Enable or disable CDP on an interface";
      }
    }
    description
      "This augment extends the configuration data of
      'Cisco-IOS-XR-ifmgr-cfg'";
  }
```

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)

Note The gRPC protocol is supported only on 64-bit platforms. The Cisco CRS series routers do not support the gRPC protocol.

The transport and encoding mechanisms for these two protocols are 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.

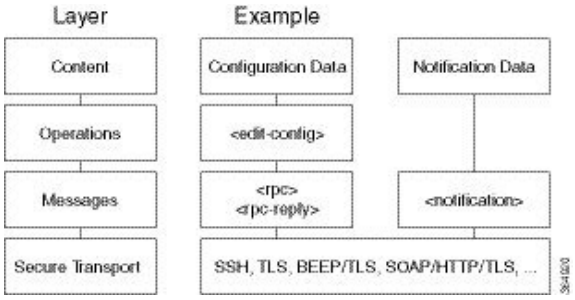
NETCONF Session

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 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.

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.
	An <unlock> operation will not succeed if either of the following conditions is true:
	• The specified lock is not currently active.
	• 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.
		`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.
		`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.
		`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>`

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`.

YANG Actions

IOS XR actions are RPC statements that trigger an operation or execute a command on the router. Theses actions are defined as YANG models using RPC statements. An action is executed when the router receives the corresponding NETCONF RPC request. Once the router executes an action, it replies with a NETCONF RPC response.

For example, **ping** command is a supported action. That means, a YANG model is defined for the **ping** command using RPC statements. This command can be executed on the router by initiating the corresponding NETCONF RPC request.

For the list of supported actions, see the following table:

Actions	YANG Models
logmsg	Cisco-IOS-XR-syslog-act
snmp	Cisco-IOS-XR-snmp-test-trap-act
rollback	Cisco-IOS-XR-cfgmgr-rollback-act
ping	Cisco-IOS-XR-ping-act Cisco-IOS-XR-ipv4-ping-act Cisco-IOS-XR-ipv6-ping-act

traceroute	Cisco-IOS-XR-traceroute-act Cisco-IOS-XR-ipv4-traceroute-act Cisco-IOS-XR-ipv6-traceroute-act
crypto	Cisco-IOS-XR-crypto-act
clear ospf	Cisco-IOS-XR-ipv4-ospf-act Cisco-IOS-XR-ipv6-ospfv3-act
clear isis	Cisco-IOS-XR-isis-act
clear bgp	Cisco-IOS-XR-ipv4-bgp-act

Example: PING NETCONF Action

This use case shows the IOS XR NETCONF action request to run the ping command on the router.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">
  <destination>
   <destination>1.2.3.4</destination>
  </destination>
 </ping>
</rpc>
```

This section shows the NETCONF action response from the router.

```
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">
  <ipv4>
   <destination>1.2.3.4</destination>
   <repeat-count>5</repeat-count>
   <data-size>100</data-size>
   <timeout>2</timeout>
   <pattern>0xabcd</pattern>
   <rotate-pattern>0</rotate-pattern>
   <reply-list>
    <result>!</result>
    <result>!</result>
    <result>!</result>
    <result>!</result>
    <result>!</result>
   </reply-list>
   <hits>5</hits>
   <total>5</total>
   <success-rate>100</success-rate>
   <rtt-min>1</rtt-min>
   <rtt-avg>1</rtt-avg>
   <rtt-max>1</rtt-max>
  </ipv4>
 </ping-response>
</rpc-reply>
```

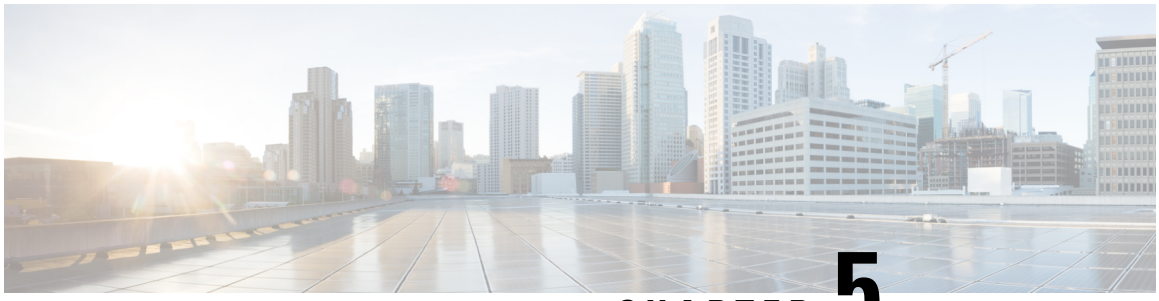
Example: XR Process Restart Action

This example shows the process restart action sent to NETCONF agent.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <sysmgr-process-restart xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-sysmgr-act">
     <process-name>processmgr</process-name>
     <location>0/RP0/CPU0</location>
  </sysmgr-process-restart>
</rpc>
```

This example shows the action response received from the NETCONF agent.

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



CHAPTER 5

Use Cases with Data Models

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

Request for AAA Access Details

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

Note If any user on XR is deleted, the local database checks whether there is a first user on Calvados VM.

- If there is a first user, no syncing occurs.
- If there is no first user, then the first user on XR (based on the order of creation) is synced to Calvados VM.

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"
  }
 }
}
```

Use OC Model to Configure Static Route

In this use case, you configure a static route with next hop using OC network instance local routing (openconfig-ni-local-routing) data model.

Note The OC interface maps all IP configurations for parent interface under a VLAN with index 0. This restricts configuring a sub interface with tag 0.

1. Create a static route with next hop interface.

```
<edit-config>
<target>
<candidate/>
</target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<network-instances xmlns="http://openconfig.net/yang/network-instance">
       <network-instance>
       <name>default</name>
        <protocols>
         <protocol>
           <identifier
xmlns:idx="http://openconfig.net/yang/policy-types">idx:STATIC</identifier>
            <name>DEFAULT</name>
          <config>
           <identifier
xmlns:idx="http://openconfig.net/yang/policy-types">idx:STATIC</identifier>
            <name>DEFAULT</name>
          </config>
<static-routes>
<static>
<prefix>5.5.5.0/24</prefix>
<config>
<prefix>5.5.5.0/24</prefix>
</config>
<next-hops>
<next-hop>
<index>link-1</index>
<config>
<index>link-1</index>
<next-hop>6.6.6.1</next-hop>
</config>
</next-hop>
</next-hops>
</static>
</static-routes>
    </protocol>
        </protocols>
       </network-instance>
      </network-instances>
</config>
</edit-config>
```

2. Verify the static route configuration.

```
<get-config>
<source>
<candidate/>
</source>
<filter>
<network-instances xmlns="http://openconfig.net/yang/network-instance">
       <network-instance>
       <name>default</name>
        <protocols>
        <protocol>
          <identifier
xmlns:idx="http://openconfig.net/yang/policy-types">idx:STATIC</identifier>
           <name>DEFAULT</name>
<static-routes>
<static>
<prefix>5.5.5.0/24</prefix>
</static>
</static-routes>
        </protocol>
        </protocols>
       </network-instance>
```

```
      </network-instances>
</filter>
</get-config>
```

Using NETCONF with Flexible CLI Configuration Groups

If you want to use NETCONF protocol with flexible CLI conifguration groups, you need to use the inherited configuration. To transition to NETCONF and YANG based configuration from a CLI configuration which includes flexible CLI configuration groups, use the following steps. Using these steps, you can retrieve all the configuration on the device which can be used in further NETCONF operations.

1. Send a NETCONF **get-config** request with source as `<running-inheritance/>` .

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

           </source>
        </get-config>
   </rpc>
   ##
   ```

   This operation returns all the configuration present on the device (inherited or expanded) in the following format:

   ```
   <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <data>
         ….
       </data>
   </rpc-reply
   ```

2. To apply the configuration to another device, send a NETCONF **edit-config** request in the following format:

   ```
   <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
     <edit-config>
       <target>
          <candidate/>
       </target>
       <config>
       </config>
     </edit-config>
   </rpc>
   ```