Programmability Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 6.5.x

First Published: 2019-01-01

Last Modified: 2019-03-29

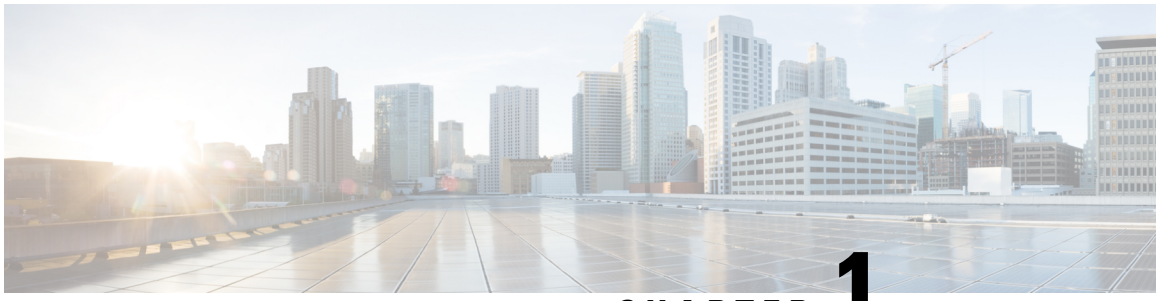
Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

- 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

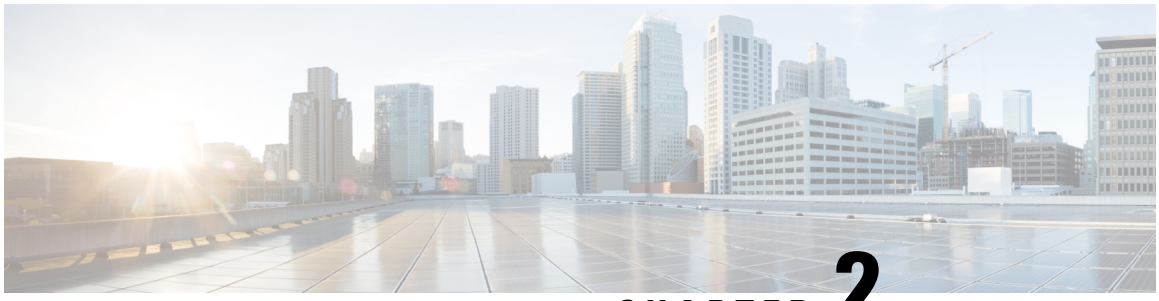
Note This product has reached end-of-life status. For more information, see the End-of-Life and End-of-Sale Notices.

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

- New and Changed Programmability Features, on page 1

New and Changed Programmability Features

Feature	Description	Changed in Release	Where Documented
Support for Open Config (OC) models: • OC NI • OC local routing • OC-MPLS • OC-RSVP-SR • OC-RPL • OC-BGP-Policy	These Cisco-supported Open Config models define YANG models for configuration and operational data.	Release 6.5.1	*Components to Use Data Models* chapter YANG Module, on page 17
Usability Enhancements for ACL YANG Models	This feature was introduced.	Release 6.5.1	*Components to Use Data Models*chapter Usability Enhancements for ACL YANG Models, on page 21
NETCONF Install YANG Actions	This feature was introduced.	Release 6.5.1	*Components to Use Data Models*chapter YANG Actions, on page 32



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) 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 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 17.

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:

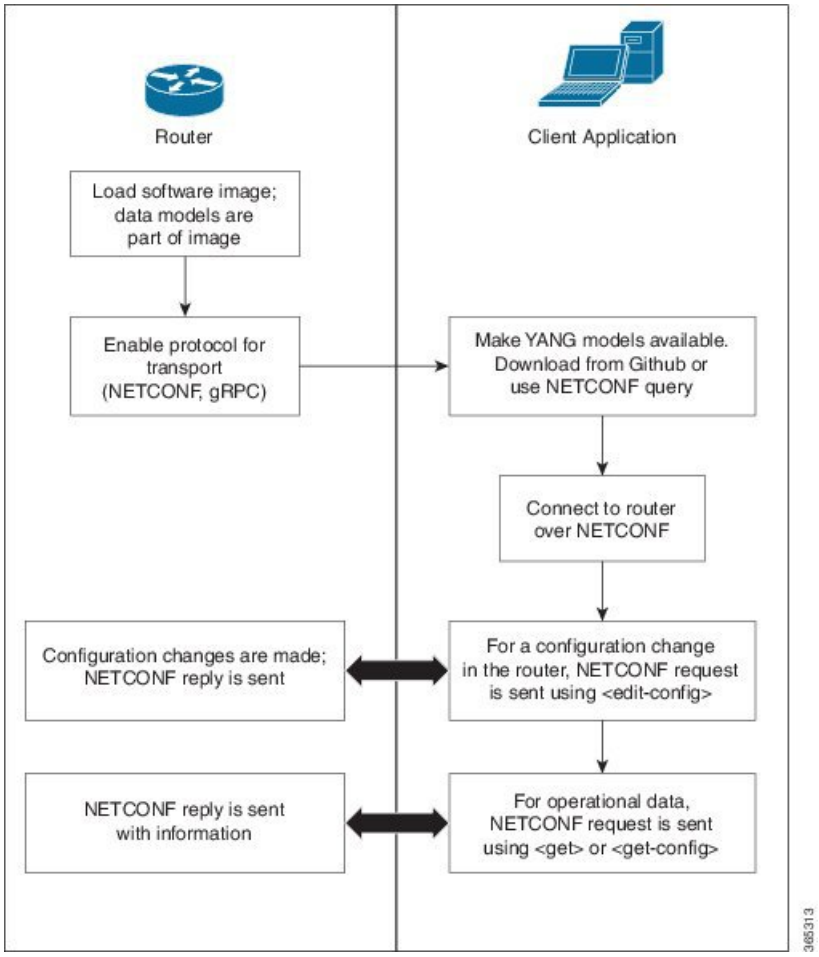
- Obtain the data models.
- Establish a connection between the router and the client using communication protocols such as NETCONF or gRPC.
- Manage the configuration of the router from the client using data models.

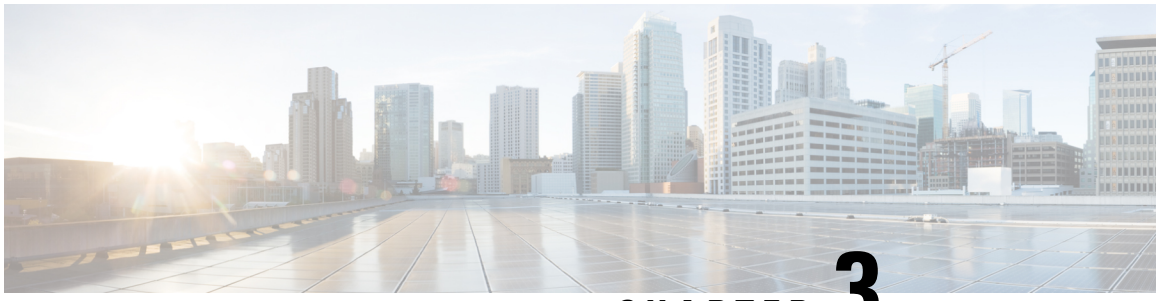
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization 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 NCS 5500 Series 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.

For installation instructions, see *Perform System Upgrade and Install Feature Packages* chapter in the System Setup and Software Installation Guide for Cisco NCS 5500 Series Routers.

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 and System Admin 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 17.

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

Note Only the first root-lr user created on XR is synchronized as the first root-system user on System Admin, while the consecutive users are not synchronized. The consecutive users created on XR do not exist in the System Admin. Hence any operations through NETCONF or gRPC that requires sysadmin access performed by the consecutive users fails. To overcome this limitation, create the user with the same name in System Admin and grant permission by assigning them to the appropriate group.

For more information about protocols, see Communication Protocols, on page 22.

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 23.

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.

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 27.

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-grpc)# port <port-number>
```

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

3. In the configuration mode, set the session parameters.

```
Router (config)# grpc{ address-family | dscp | max-request-per-user | max-request-total
 | max-streams | max-streams-per-user | no-tls | service-layer | tls-cipher | tls-mutual
 | tls-trustpoint | vrf }
```

where:

- **address-family:** set the address family identifier type
- **dscp:** set QoS marking DSCP on transmitted gRPC
- **max-request-per-user:** set the maximum concurrent requests per user
- **max-request-total:** set the maximum concurrent requests in total
- **max-streams:** set the maximum number of concurrent gRPC requests. The maximum subscription limit is 128 requests. The default is 32 requests
- **max-streams-per-user:** set the maximum concurrent gRPC requests for each user. The maximum subscription limit is 128 requests. The default is 32 requests
- **no-tls:** disable transport layer security (TLS). The TLS is enabled by default.

- **service-layer:** enable the grpc service layer configuration
- **tls-cipher:** enable the gRPC TLS cipher suites
- **tls-mutual:** set the mutual authentication
- **tls-trustpoint:** configure trustpoint
- **server-vrf:** enable server vrf

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 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 20.

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

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 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.

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.

Commit Configuration

Commit the configuration to set the new values in the current running configuration.

The configuration can also be committed through a `confimed-commit` operation. NETCONF and gRPC supports confirmed-commit RPC. This RPC requires an explicit confirmation from the user before the configuration takes effect on the router. This feature helps in verifying that the change in the configuration works correctly and does not cause fluctuation in the management connectivity. If the configuration change causes loss of management connectivity, the configuration is automatically rolled back to the previous committed configuration after the default `confirm-timeout` period of 600 seconds.

To commit a configuration, use </commit> RPC:

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

To confirm-commit a configuration:

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

The confirmed-commit capability supports the <cancel-commit> operation and the <confirmed>, <confirm-timeout>, <persist>, and <persist-id> parameters for the <commit> operation.

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

```
      <commit>
        <confirmed/>
        <persist>IQ,d4668</persist>
        <confirm-timeout>120</confirm-timeout>
      </commit>
</rpc>
```

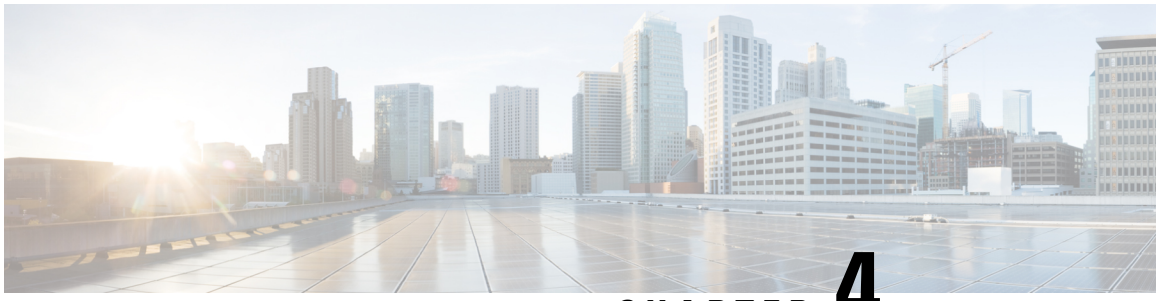
A confirmed-commit request will fail with `Datastore Locked` error if:

- another operation is performed between a confirmed-commit and a confirming-commit operation
- another session has an active confirmed-commit request and a persist ID was not provided
- a persist ID was provided but did not match the persist ID of the active confirmed-commit session

gRPC Confirmed-Commit for Merging Configuration

gRPC confirmed-commit request can be issued for exisiting merge-config and cli-config operations. In this example, the request is made for merge-cli operation.

```
manageability/ems/client/client -oper merge-config -server_addr="<address>" -json_in_file
<directory-path>/<file>.json
-confirmed=yes -confirm_timeout=400
enter PID:14917:main.main
emsMergeConfig: Received ReqId 14917, Response '
----------------- gRPC Summary ----------------------
Operation: merge-config
Number of iterations: 1
Total bytes transferred: 126
Number of bytes per second: 374
Round trip throughputs Mbps: 0.002999
Ave elapsed time in seconds: 0.336079
Min elapsed time in seconds: 0.336079
Max elapsed time in seconds: 0.336079
--------------- End gRPC Summary ---------------------
The confirmed commit request should be followed by a confirming commit to make the
configuration permanent:
manageability/ems/client/client -oper commit -server_addr="<address>"
enter PID:14917:main.main
emsCommitConfig: Received ReqId 14917, Response '
----------------- gRPC Summary ----------------------
Operation: commit
Number of iterations: 1
Total bytes transferred: 126
Number of bytes per second: 374
Round trip throughputs Mbps: 0.002999
Ave elapsed time in seconds: 0.336079
Min elapsed time in seconds: 0.336079
Max elapsed time in seconds: 0.336079
--------------- End gRPC Summary ---------------------
```



CHAPTER 4

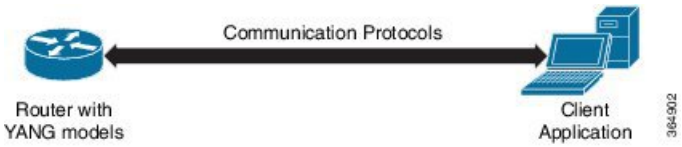
Components to Use Data Models

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 17
- Communication Protocols, on page 22
- YANG Actions, on page 32

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 and gRPC-enabled 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.

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.

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)
```

Usability Enhancements for ACL YANG Models

This feature addresses some of the issues identified with native ACL YANG models that affect usability of the YANG model. It improves user-friendliness and standards compliance in the following native ACL YANG models:

- Cisco-IOS-XR-es-acl-cfg
- Cisco-IOS-XR-ipv4-acl-cfg
- Cisco-IOS-XR-ipv6-acl-cfg

The following issues are addressed as part of this enhancement:

- Lack of revision dates and descriptions

  Problem: When the revision changed on an ACL-model YANG file, the changes associated with the new revision were not accurately described.

  Solution: Includes description for the changes made in each subsequent version without removing the information for the previous versions.

  Example from Cisco-IOS-XR-ipv4-acl-cfg.yang :

```
revision "2018-04-03" {
   description
     "6.5.1 revision. Correct enum value for Next-hop-type.";
 }

 revision "2018-03-23" {
   description
     "6.5.1 revision. Removing none-next-type.";
 }
```

- Unconstrained use of strings

  Problem: ACL native models use leaves defined as *string* type; however, the string length is undefined or incorrect.

  Solution: Pattern and length checking is added to leaves using *string* types. This allows NETCONF to control these checks, rather than relying on ACL verifier at commit time.

  In the following example, the length of the string is limited to 255 characters and only alphanumeric characters are allowed.

```
typedef my-base-str-type { type string {
length "1..255";
pattern "[0-9a-fA-F]*";
} }
```

- Lack of good description statements

  Problem: Most leaves in the native ACL models had description fields that did not have good description, which impacts the ease of use and understanding of the YANG model.

  Solution: Leaf description fields are updated to provide useful information.

- Lack of verification and inconsistent behavior across ACL-range leaves

  Problem: YANG model supports different combinations of leaves for containers with upper value, lower value and operator leaves. As a result, some of the configurations can be invalid from the CLI perspective.

  Solution: The native ACL models are improved to include various combinations of leaves that are supported by the YANG model. Also, the parsing behavior across all ACL-range leaves are aligned to remain consistent.

- Inconsistency between protocol-operator input and output

  Problem: NETCONF output should be consistent with user input.

  Solution: The protocol-operator leaf is made as non-mandatory, and need not be set to *equal*.

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. 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.

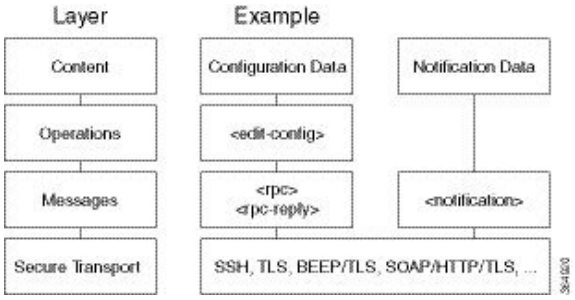
NETCONF Session

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>`

Note The System admin models support only `<get>` and `<get-config>` operations. The `<edit-config>` operation works only with the `merge` operation. The other operations such as `<delete>`, `<remove>`, `<replace>` and so on are not supported.

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) {};

    rpc CommitReplace(CommitReplaceArgs) returns(CommitReplaceReply) {};
}
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 CommitReplaceArgs {
     int64 ReqId = 1;
     string cli = 2;
     string yangjson = 3;
}

message CommitReplaceReply {
     int64 ResReqId = 1;
     string errors = 2;
}
```

Example for gRPCExec configuration:

```
service gRPCExec {
    rpc ShowCmdTextOutput(ShowCmdArgs) returns(stream ShowCmdTextReply) {};
    rpc ShowCmdJSONOutput(ShowCmdArgs) returns(stream ShowCmdJSONReply) {};
    rpc ActionJSON(ActionJSONArgs) returns(stream ActionJSONReply) {};
}

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

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

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

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

Example for OpenConfiggRPC configuration:

```
service OpenConfiggRPC {
     rpc SubscribeTelemetry(SubscribeRequest) returns (stream SubscribeResponse) {};
     rpc UnSubscribeTelemetry(CancelSubscribeReq) returns (SubscribeResponse) {};
     rpc GetModels(GetModelsInput) returns (GetModelsOutput) {};
}

message GetModelsInput {
     uint64  requestId   = 1;
     string  name        = 2;
    string  namespace   = 3;
    string  version     = 4;
```

```
    enum MODLE_REQUEST_TYPE {
         SUMMARY = 0;
         DETAIL  = 1;
     }
     MODLE_REQUEST_TYPE requestType = 5;
}

message GetModelsOutput {
     uint64  requestId   = 1;
     message ModelInfo {
         string  name        = 1;
         string  namespace   = 2;
        string  version     = 3;
         GET_MODEL_TYPE  modelType = 4;
         string modelData = 5;
     }
     repeated ModelInfo models = 2;
     OC_RPC_RESPONSE_TYPE responseCode = 3;
     string msg = 4;
}
```

gRPC Operations

The gRPC operations include:

gRPC Operation	Description
GetConfig	Retrieves a configuration
GetModels	Gets the supported Yang models on the router
MergeConfig	Appends to an existing configuration
DeleteConfig	Deletes a configuration
ReplaceConfig	Modifies a part of an existing configuration
CommitReplace	Replaces existing configuration with the new configuration file provided
GetOper	Gets operational data using JSON
CliConfig	Invokes the CLI configuration
ShowCmdTextOutput	Displays the output of show command
ShowCmdJSONOutput	Displays the JSON output of show command
ActionJSON	Displays the gRPC JSON action

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]
}
```

gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

gNNMI uses gRPC as the transport protocol and the configuration is same as that of gRPC. These gNMI RPCs are supported:

gNMI RPC	gNMI RPC Request	Description
Capabilities		Initial handshake between the network device (server) and the client to exchange capability information such as supported data models
Set	`message SetRequest {}`	Modifies data associated with a model on a network device from a client
Get	`message GetRequest {}`	Retrieves data from a network device
Subscribe	`message SubscribeRequest {}`	Control data subscriptions on server

For more information about gNMI, see Github repository.

YANG Actions

IOS XR and System Admin 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 or gRPC request. Once the router executes an action, it replies with a NETCONF RPC or gRPC 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 or gRPC request.

Note NETCONF supports XML format, and gRPC supports JSON format.

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
System Process Mgmt : process (restart)	Cisco-IOS-XR-sysmgr-act Cisco-IOS-XR-sysadmin-pm
System Process Mgmt : Reload (System Admin virtual machine (VM) reload, line card (LC) reload)	Cisco-IOS-XR-sysadmin-sm
System Process Mgmt : Reload (IOS XR VM node reload from System Admin)	Cisco-IOS-XR-sysadmin-sdr-mgr
System Process Mgmt : Install	Cisco-IOS-XR-spirit-install-act
dumpcore	Cisco-IOS-XR-spirit-corehelper-cfg

Note The System admin models support only `<get>` and `<get-config>` operations. The `<edit-config>` operation works only with the `merge` operation. The other operations such as `<delete>`, `<remove>`, `<replace>` and so on are not supported.

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>
```

Example: Shutdown Dumper Process

This use case shows the System Admin NETCONF action request to shut down dumper process on the router.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<action xmlns="http://tail-f.com/ns/netconf/actions/1.0">
   <data>
     <processes xmlns="http://www.cisco.com/ns/yang/Cisco-IOS-XR-sysadmin-pm">
       <all-locations>
         <location>0/RP0</location>
         <name>
           <proc-name>dumper</proc-name>
           <instance-id>0</instance-id>
           <proc-action>
             <do-what>shutdown</do-what>
             <user-name>root</user-name>
             <user-ip>1.2.3.4</user-ip>
           </proc-action>
         </name>
       </all-locations>
     </processes>
   </data>
```

```
</action>
</rpc>
```

This section shows the NETCONF action response from the router.

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <processes xmlns="http://www.cisco.com/ns/yang/Cisco-IOS-XR-sysadmin-pm">
   <all-locations>
    <location>0/RP0</location>
    <name>
     <proc-name>dumper</proc-name>
     <instance-id>0</instance-id>
     <proc-action>
      <proc-action-status>User root (1.2.3.4) requested shutdown for process dumper(0) at
0/RP0
 'Sending signal 15 to stop process dumper(IID 0) pid=2439'</proc-action-status>
     </proc-action>
    </name>
   </all-locations>
  </processes>
 </data>
</rpc-reply>
```

Example: Copy Action

This example shows the RPC request and response for `copy` action:

RPC request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <copy xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">
    <sourcename>//root:<location>/100MB.txt</sourcename>
    <destinationname>/</destinationname>
    <sourcefilesystem>ftp:</sourcefilesystem>
    <destinationfilesystem>harddisk:</destinationfilesystem>
    <destinationlocation>0/RSP1/CPU0</destinationlocation>
  </copy>
</rpc>
```

RPC response:

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">Successfully
completed copy operation</response>
</rpc-reply>

8.261830565s elapsed
```

Example: Delete Action

This example shows the RPC request and response for `delete` action:

RPC request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<delete xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">
    <name>harddisk:/netconf.txt</name>
```

```
  </delete>
</rpc>
```

RPC response:

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">Successfully
 completed delete operation</response>
</rpc-reply>

395.099948ms elapsed
```

Example: Install Action

This example shows the Install action request sent to NETCONF agent.

```
  <install-add xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-spirit-install-act">
      <packagepath>/nobackup/hanaik/yang_project/img-xrv9k</packagepath>
      <packagename>xrv9k-mpls-2.1.0.0-r64102I.x86_64.rpm</packagename>
  </install-add>
```

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

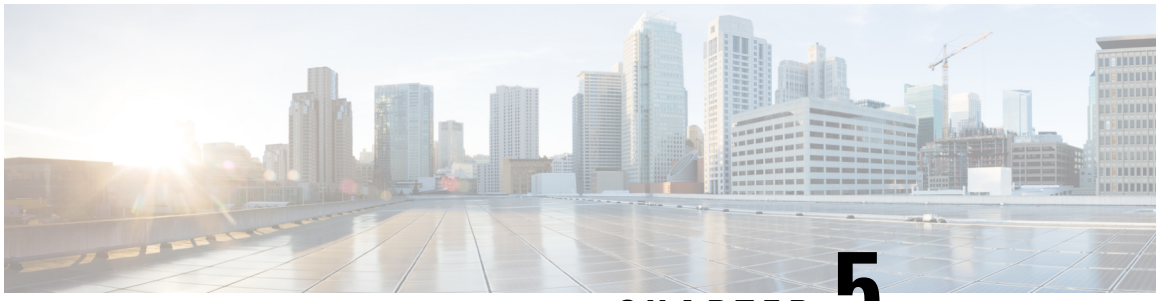
```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <op-id xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-spirit-install-act">6</op-id>
</rpc-reply>
```

This example shows how to use *install add rpc* request with multiple packages enclosed within *packagename* tag.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<install-add xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-spirit-install-act">
   <packagepath>http://10.105.227.154/install_repo/fretta/651/651_02</packagepath>
        <packagename>ncs5500-k9sec-3.1.0.0-r65102I.x86_64.rpm</packagename>
        <packagename>ncs5500-li-1.0.0.0-r65102I.x86_64.rpm</packagename>
        <packagename>ncs5500-mcast-2.1.0.0-r65102I.x86_64.rpm</packagename>
        <packagename>ncs5500-mini-x.iso-6.5.1.02I</packagename>
        <packagename>ncs5500-mpls-2.1.0.0-r65102I.x86_64.rpm</packagename>
</install-add>
</rpc>
```

Restrictions for Install Action

- **Install upgrade** command is deprecated. Hence, use **install update** command instead of the **install upgrade** command.
- Only one request can be sent at a time.
- ISSU is not supported.
- Install Yang using NETCONF action can accept a maximum of 32 input parameters. Input parameters can be any inputs used in **install action** commands, such as package names to add, activate, deactivate, or remove, and operation IDs to retrieve any particular log related to that operation.



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>
   ```