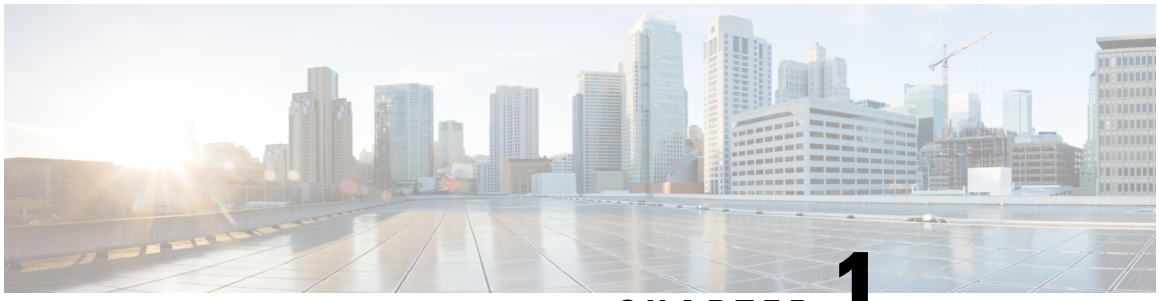
Programmability Configuration Guide for Cisco NCS 4000 Series Routers, IOS XR Release 6.5.x

First Published: 2021-08-30

Last Modified: 2022-12-08

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

CONTENTS



CHAPTER 1

Drive Network Automation Using Programmable YANG Data Models

Table 1: Feature History

Feature Name	Release Information	Feature Description
NCS 4000 YANG Data models	Cisco IOS XR Release 6.5.31	YANG data models are supported on NCS 4000 instead of CLI commands. The data models are written in an industry-defined language and is used to automate configuration tasks and retrieve operational data across heterogeneous devices in a network. Using model-driven programmability results in scalability.

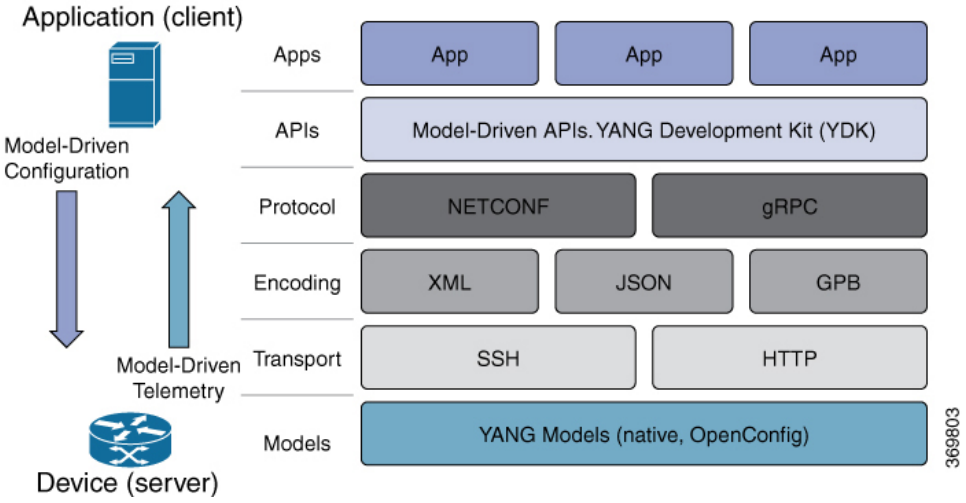
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the 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, you need an automated mechanism to manage your network.

Cisco IOS XR supports a programmatic way of configuring and collecting operational data of a network device using data models. They replace the process of manual configuration, which is proprietary, and highly text-based. The data models are written in an industry-defined language and is used to automate configuration task and retreive operational data across heterogeneous devices in a network. Although configurations using CLIs are easier and human-readable, automating the configuration using model-driven programmability results in scalability.

Model-driven programmability provides a simple, flexible and rich framework for device programmability. This programmability framework provides multiple choices to interface with an IOS XR device in terms of transport, protocol and encoding. These choices are decoupled from the models for greater flexibility.

The following image shows the layers in model-driven programmability:

Figure 1: Model-driven Programmability Layers



Data models provides access to the capabilities of the devices in a network using Network Configuration Protocol. The operations on the router are carried out by the protocols using YANG models to automate and programme operations in a network.

Benefits of Data Models

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.

This article describes how you benefit from using data models to programmatically manage your network operations.

- YANG Data Model, on page 2
- Access the Data Models, on page 4
- Communication Protocols, on page 6

YANG Data Model

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 enabled applications.

YANG models can be:

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

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 JSON data is based on YANG module name and not YANG namespace.

Structure of YANG Data Model

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.

For more information about Data Models, see RFC 6244.

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

Access the Data Models

You can access the Cisco IOS XR native data models from GitHub, a software development platform that provides hosting services for version control.

You can also access the supported data models from the router. The router ships with the YANG files that define the data models. Use NETCONF protocol to view the data models available on the router using `ietf-netconf-monitoring` request.

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

All the supported YANG models are displayed as response to the RPC request.

```
<rpc-reply message-id="urn:uuid:74c81086-9706-4d59-b50e-25dc1a627c6b"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="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-ppp-ea-oper</identifier>
          <version>2015-11-09</version>
         <format>yang</format>
          <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ppp-ea-oper</namespace>
          <location>NETCONF</location>
        </schema>

        <schema>
          <identifier>Cisco-IOS-XR-ppp-ea-oper-sub1</identifier>
          <version>2015-11-09</version>
          <format>yang</format>
          <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ppp-ea-oper</namespace>
```

```
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-rsvp-oper</identifier>
        <version>2017-09-07</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-rsvp-oper</namespace>
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-rsvp-oper-sub1</identifier>
        <version>2017-09-07</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-rsvp-oper</namespace>
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-udp-oper</identifier>
        <version>2018-03-04</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-udp-oper</namespace>
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-udp-oper-sub4</identifier>
        <version>2018-03-04</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-udp-oper</namespace>
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-udp-oper-sub1</identifier>
        <version>2018-03-04</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-udp-oper</namespace>
        <location>NETCONF</location>
      </schema>

      <schema>
        <identifier>Cisco-IOS-XR-ip-udp-oper-sub3</identifier>
        <version>2018-03-04</version>
        <format>yang</format>
        <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-ip-udp-oper</namespace>
        <location>NETCONF</location>
      </schema>
            .
            .

      </schema>
        <identifier>openconfig-bgp-policy</identifier>
        <version>2017-02-02</version>
        <format>yang</format>
        <namespace>http://openconfig.net/yang/bgp-policy</namespace>
        <location>NETCONF</location>
      </schema>
    </schemas>
  </netconf-state>
 </data>
</rpc-reply>
```

The models are in the `.yang` format. A model with:

- `-oper` in the model name indicates an operational model. For example, `Cisco-IOS-XR-invmgr-oper.yang` is an operational model for inventory data.
- `-cfg` indicates a configuration model. For example, `Cisco-IOS-XR-controller-OTU-cfg.yang` is a configuration model for OTU.

Communication Protocols

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

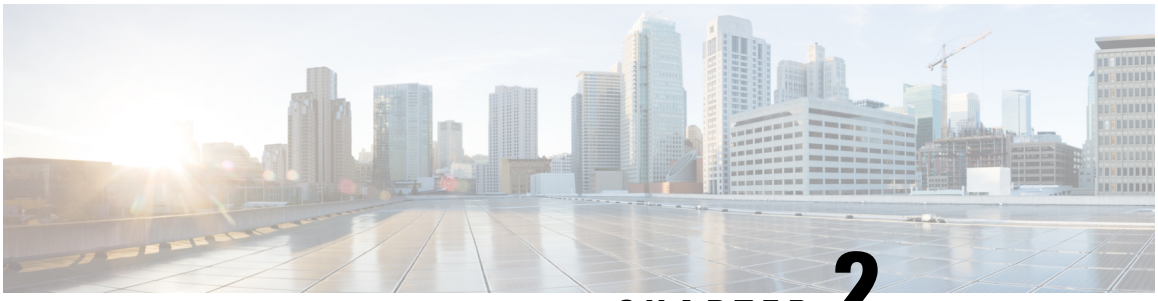
YANG uses the Network Configuration Protocol (NETCONF) .

The transport and encoding mechanisms for theis protocol are shown in the table:

Protocol	Transport	Encoding/ Decoding
NETCONF	ssh	xml
	tcp	json

NETCONF Protocol

NETCONF provides mechanisms to install, manipulate, or delete the configuration on network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. You use a simple NETCONF RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. To get started with issuing NETCONF RPCs to configure network features using data models, see .



CHAPTER 2

Use NETCONF Protocol to Define Network Operations with Data Models

XR devices ship with the YANG files that define the data models they support. Using a management protocol such as NETCONF, you can programmatically query a device for the list of models it supports and retrieve the model files.

Network Configuration Protocol (NETCONF) is a standard transport protocol that communicates with network devices. NETCONF provides mechanisms to edit configuration data and retrieve operational data from network devices. The configuration data represents the way interfaces, routing protocols and other network features are provisioned. The operational data represents the interface statistics, memory utilization, errors, and so on.

NETCONF uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. It 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 defines how to communicate with the devices, but does not handle what data is exchanged between the client and the server.

To enable NETCONF, run the following commands:

```
RP/0/RP0:ios#configure
Sun Jun 21 22:32:45.159 IST
RP/0/RP0:ios(config)#netconf agent tty
RP/0/RP0:ios(config-netconf-tty)#netconf-yang agent
RP/0/RP0:ios(config-ncy-agent)#ssh server netconf vrf default
RP/0/RP0:ios(config)#commit
Sun Jun 21 22:33:07.995 IST
RP/0/RP0:ios(config)#end
```

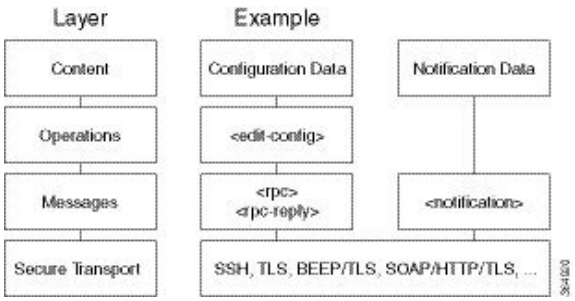
NETCONF Session

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 protocol can be partitioned into four layers:

Figure 2: 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.

This article describes, with a use case to configure the local time on a router, how data models help in a faster programmatic configuration as comapared to CLI.

- NETCONF Operations, on page 8
- Configure an Interface Port Mode Using Data Model in a NETCONF Session, on page 12
- Configure Breakouts Using Data Model in a NETCONF Session, on page 15

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
```

These NETCONF operations are described in the following table:

NETCONF Operation	Description	Example

<get-config>	Retrieves all or part of a specified configuration from a named data store	Retrieves the nsf configuration using Cisco-IOS-XR-clns-isis-cfg.
		`<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:0459351e-d60f-4c6d-a195-ffee7efb3921">`
		`<get-config>`
		`<source>`
		`<running/>`
		`</source>`
		`<filter>`
		`<isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-cfg">`
		`<instances>`
		`<instance>`
		`<nsf/>`
		`</instance>`
		`</instances>`
		`</isis>`
		`</filter>`
		`</get-config>`
		`</rpc>`

<get>	Retrieves running configuration and device state information	Retrieves the optics configuration on particular interface.
		`<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">`
		`<get>`
		`<filter>`
		`<optics-oper xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-oper">`
		`<optics-ports>`
		`<optics-port>`
		`<name>Optics0/2/0/11</name>`
		`<optics-db-info/>`
		`</optics-port>`
		`</optics-ports>`
		`</optics-oper>`
		`</filter>`
		`</get>`
		`</rpc>`

NETCONF Operation	Description	Example

<edit-config>	Loads all or part of a specified configuration to the specified target configuration	Edits the nsf configuration using Cisco-IOS-XR-clns-isis-cfg
		`<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:68baa1ad-d574-4530-ac65-fd83c8cbf2ea">`
		`<edit-config>`
		`<target>`
		`<candidate/>`
		`</target>`
		`<config>`
		`<isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-cfg">`
		`<instances>`
		`<instance>`
		`<instance-name>100</instance-name>`
		`<nsf>`
		`<flavor>cisco-proprietary-nsf</flavor>`
		`</nsf>`
		`</instance>`
		`</instances>`
		`</isis>`
		`</config>`
		`</edit-config>`
		`</rpc>`
		`<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">`
		`<commit/>`
		`</rpc>`

NETCONF Operation to Get Configuration

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`.

This example shows how a NETCONF <get-config> and <edit-config> request works for the ISIS configuration.

To retrieve the nsf configuration using Cisco-IOS-XR-clns-isis-cfg:

Netconf Request (Client to Router)	Netconf Response (Router to Client)
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:0459351e-d60f-4c6d-a195-ffee7efb3921"> <get-config> <source> <running/> </source> <filter> <isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-cfg"> <instances> <instance> <nsf/> </instance> </instances> </isis> </filter> </get-config> </rpc>	<rpc-reply message-id="urn:uuid:0459351e-d60f-4c6d-a195-ffee7efb3921" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <data> <isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-cfg"> <instances> <instance> <instance-name>100</instance-name> <nsf> <**flavor>ietf-standard-nsf</flavor>** </nsf> </instance> </instances> </isis> </data> </rpc-reply>

The following displays the command output from the router:

```
RP/0/RP0:ios#show run router isis
Fri Jul  3 22:22:22.150 IST
router isis 100
is-type level-2-only
net 49.0000.0000.0000.0001.00
nsr
nsf ietf
address-family ipv4 unicast
  metric-style wide
  mpls traffic-eng level-2-only
  mpls traffic-eng router-id Loopback1
```

To edit the nsf configuration, using Cisco-IOS-XR-clns-isis-cfg:

Netconf Request (Client to Router)	Netconf Response (Router to Client)
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:68baa1ad-d574-4530-ac65-fd83c8cbf2ea"> <edit-config> <target> <candidate/> </target> <config> <isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-cfg"> <instances> <instance> <instance-name>100</instance-name> <nsf> <flavor>cisco-proprietary-nsf</flavor> </nsf> </instance> </instances> </isis> </config> </edit-config> </rpc> <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102"> <commit/> </rpc>	<?xml version="1.0" ?> <rpc-reply message-id="urn:uuid:68baa1ad-d574-4530-ac65-fd83c8cbf2ea" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>

After a successful edit config operation, the router configuration is changed to:

```
RP/0/RP0:ios#show run router isis
Fri Jul  3 22:25:33.793 IST
router isis 100
is-type level-2-only
net 49.0000.0000.0000.0001.00
nsr
nsf cisco
address-family ipv4 unicast
  metric-style wide
  mpls traffic-eng level-2-only
  mpls traffic-eng router-id Loopback1
```

Configure an Interface Port Mode Using Data Model in a NETCONF Session

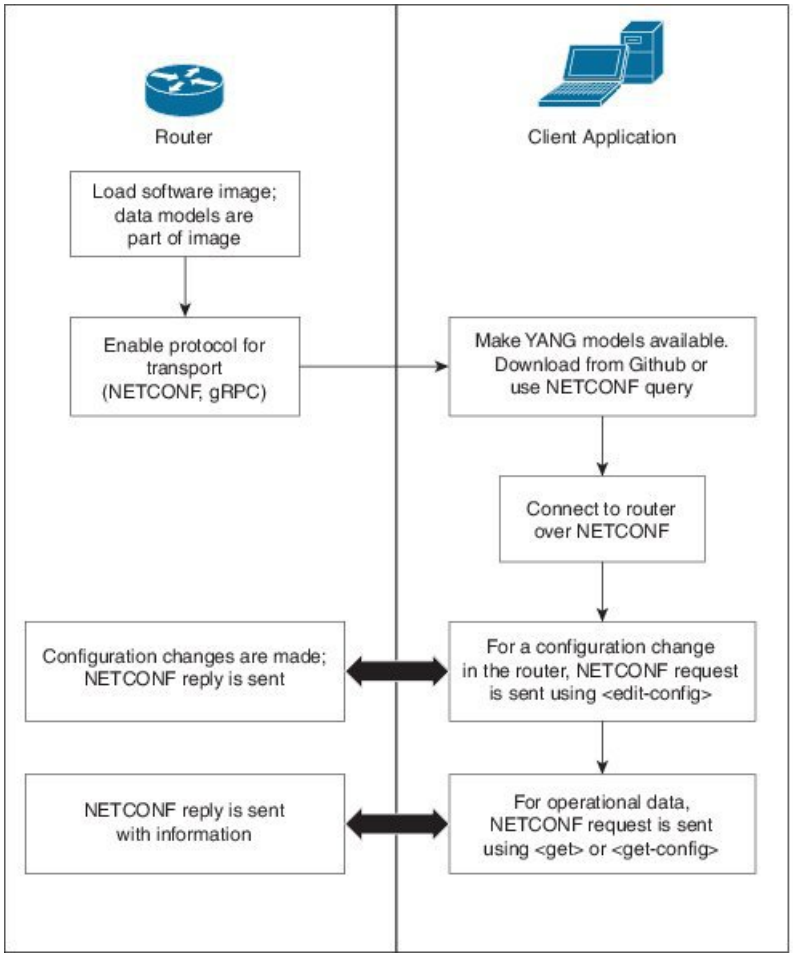
NETCONF is an XML-based protocol used over Secure Shell (SSH) transport to configure a network. The client applications use this protocol to request information from the router, and make configuration changes to the router.

The process for using data models involves:

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

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

Figure 3: Process for Using Data Models



In this section, you use the native data model `Cisco-IOS-XR-portmode-cfg.yang` to programmatically configure and verify the port mode of an interface using a NETCONF session.

Procedure

Step 1 Explore the native configuration model for the port mode configuration on the router.

Example:

```
RP/0/RP0:ios#show netconf-yang capabilities | in "Cisco-IOS-XR-portmode-cfg"
Mon Aug 24 15:34:14.483 IST
http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg                    |2020-02-24|
RP/0/RP0:ios#
```

Step 2 Explore the native configuration model on the NETCONF client.

The `Cisco-IOS-XR-portmode-cfg` is present within `Cisco-IOS-XR-ifmgr-cfg`. When you load `Cisco-IOS-XR-portmode-cfg`, the `Cisco-IOS-XR-ifmgr-cfg` automatically loads and you can find the `Cisco-IOS-XR-portmode-cfg` when you expand the `Cisco-IOS-XR-ifmgr-cfg`.

Step 3 Retrieve the current port mode configuration on the router.

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:7d7b08bb-a688-4b4a-bf76-b9367bfaa150"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
      <interface-configuration>
        <active>act</active>
        <interface-name>Optics0/3/0/6</interface-name>
        <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg">
          <info>
            <rate>none</rate>
            <mapping>none</mapping>
            <framing>opu3</framing>
            <type>otn</type>
          </info>
        </port-mode>
      </interface-configuration>
    </interface-configurations>
  </data>
</rpc-reply>
```

Step 4 Change the port mode configuration using the `edit-config` option.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
        <interface-configuration>
          <active>act</active>
          <interface-name>Optics0/3/0/6</interface-name>
          <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg">
            <info>
              <rate>none</rate>
              <mapping>none</mapping>
              <framing>opu2</framing>
              <type>otn</type>
            </info>
          </port-mode>
        </interface-configuration>
      </interface-configurations>
    </config>
  </edit-config>
</rpc>
```

Step 5 View the updated port mode configuration of the interface.

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:a122c79d-1cb0-4a02-b4e2-3cc5c7efef9a"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
```

```
    <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
      <interface-configuration>
        <active>act</active>
        <interface-name>Optics0/3/0/6</interface-name>
        <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg">
          <info>
            <rate>none</rate>
            <mapping>none</mapping>
            <framing>opu2</framing>
            <type>otn</type>
          </info>
        </port-mode>
      </interface-configuration>
   </interface-configurations>
  </data>
</rpc-reply>
```

Configure Breakouts Using Data Model in a NETCONF Session

In this section, you use the native data model `Cisco-IOS-XR-portmode-cfg.yang` to programmatically configure the breakout for an interface using a NETCONF session.

Procedure

Step 1 Explore the native configuration model for the port mode configuration on the router.

Example:

```
RP/0/RP0:ios#show netconf-yang capabilities | in "Cisco-IOS-XR-portmode-cfg"
Mon Aug 24 15:34:14.483 IST
http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg                              |2020-02-24|
RP/0/RP0:ios#
```

Step 2 Explore the native configuration model on the NETCONF client.

The `Cisco-IOS-XR-portmode-cfg` is present within `Cisco-IOS-XR-ifmgr-cfg`. When you load `Cisco-IOS-XR-portmode-cfg`, the `Cisco-IOS-XR-ifmgr-cfg` automatically loads and you can find the `Cisco-IOS-XR-portmode-cfg` when you expand the `Cisco-IOS-XR-ifmgr-cfg`.

Step 3 To configure the breakout, use the `edit-config` option.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target><candidate/></target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
    <active>act</active>
    <interface-name>Optics0/7/0/0</interface-name>
    <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg">
     <lane-numbers>
      <lane-number>
       <lane-no>1</lane-no>
       <rate>10ge</rate>
       <mapping>none</mapping>
```

```
       <framing>packet</framing>
       <type>ethernet</type>
      </lane-number>
     </lane-numbers>
    </port-mode>
   </interface-configuration>
   </interface-configurations>
 </config>
</edit-config>
</rpc>
```

Note When you configure breakouts for ethernet packets through NETCONF, you must explicitly define the rate as "10GE" and the mapping as "None".

Step 4 To view the updated port mode configuration of the interface, use the `get` option.

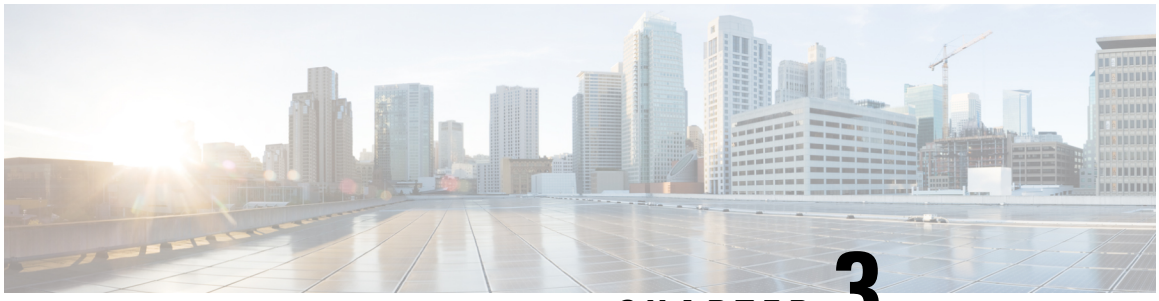
```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get>
<filter>
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
      <interface-name>Optics0/7/0/0</interface-name>
   </interface-configuration>
    </interface-configurations>
    </filter>
</get>

 </rpc>
```

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
    <active>act</active>
    <interface-name>Optics0/7/0/0</interface-name>
    <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg">
     <lane-numbers>
      <lane-number>
       <lane-no>1</lane-no>
       <rate>10ge</rate>
       <mapping>none</mapping>
       <framing>packet</framing>
       <type>ethernet</type>
      </lane-number>
     </lane-numbers>
    </port-mode>
   </interface-configuration>
  </interface-configurations>
#22
 </data>
</rpc-reply>
```

Step 5 (Optional) To delete the breakout, use the `edit-config` option.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target><candidate/></target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
    <active>act</active>
    <interface-name>Optics0/7/0/0</interface-name>
    <port-mode xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-portmode-cfg"
```

```
xc:operation="delete">
     <lane-numbers>
      <lane-number>
       <lane-no>1</lane-no>
       <rate>10ge</rate>
       <mapping>none</mapping>
       <framing>packet</framing>
       <type>ethernet</type>
      </lane-number>
     </lane-numbers>
    </port-mode>
   </interface-configuration>
   </interface-configurations>
 </config>
</edit-config>
</rpc>
```



CHAPTER 3

Supported YANG Models

YANG Models

Table 2: Feature History

Feature Name	Release Information	Feature Description
YANG Models	Cisco IOS XR Release 6.5.32	The following YANG models are introduced in this release: • Ping • Traceroute • CFM • CFM Oper Data • OBFL Data Clear • Slice Control

The following table contains the YANG models created for NCS 4000.

Functionality	YANG Models
Port Mode Config	Cisco-IOS-controller-optics-cfg
Port Mode Config	Cisco-IOS-XR-ifmgr-cfg
Optics PM Config	Cisco-IOS-XR-pmengine-cfg
Get Optics Oper Data	Cisco-IOS-XR-controller-optics-oper
OTU Config	Cisco-IOS-XR-controller-OTU-cfg
ODU Config	Cisco-IOS-XR-controller-odu-cfg
OTU Oper Data	Cisco-IOS-XR-controller-otu-oper

Functionality	YANG Models
ODU Oper Data	Cisco-IOS-XR-controller-odu-oper
GCC Config	Cisco-IOS-XR-ipv4-io-cfg
GCC Oper Data	Cisco-IOS-XR-ipv4-io-oper
OTN PM Config	Cisco-IOS-XR-pmengine-cfg
OTN PM Oper Data	Cisco-IOS-XR-pmengine-oper
Packet Interface Config	Cisco-IOS-XR-ifmgr-cfg
	Cisco-IOS-XR-infra-statsd-cfg
	Cisco-IOS-XR-ncs4k-freqsync-cfg
	Cisco-IOS-XR-l2-eth-infra-cfg
	Cisco-IOS-XR-qos-ma-cfg
	Cisco-IOS-XR-ipv4-io-cfg
Layer2 and Layer3 Bundle Config	Cisco-IOS-XR-bundlemgr-cfg
Policy-Map Config	Cisco-IOS-XR-infra-policymgr-cfg
CFM Config	Cisco-IOS-XR-ethernet-cfm-cfg
ISIS Config	Cisco-IOS-XR-clns-isis-cfg
BGP Config	Cisco-IOS-XR-ipv4-bgp-cfg
Pseudowire (PW) Class Config	Cisco-IOS-XR-l2vpn-cfg
LMP Config	Cisco-IOS-XR-lmp-cfg
Explicit Path Config	Cisco-IOS-XR-ip-iep-cfg
GMPLS Optical UNI Config	Cisco-IOS-XR-mpls-te-cfg
GMPLS Optical NNI Config	Cisco-IOS-XR-mpls-te-cfg
Route Policy Config	Cisco-IOS-XR-policy-repository-cfg
OSPF Config	Cisco-IOS-XR-ipv4-ospf-cfg
MPLS-TE Config	Cisco-IOS-XR-mpls-te-cfg
RSVP Config	Cisco-IOS-XR-ip-rsvp-cfg
Ethernet Interface Oper Data	Cisco-IOS-XR-l2-eth-infra-oper
QOS Oper Data	Cisco-IOS-XR-qos-ma-oper
Bundle Oper Data	Cisco-IOS-XR-bundlemgr-oper

Functionality	YANG Models
Route Policy Oper Data	Cisco-IOS-XR-infra-policymgr-oper
ISIS Oper Data	Cisco-IOS-XR-clns-isis-oper
BGP Oper Data	Cisco-IOS-XR-ipv4-bgp-oper
PW Oper Data	Cisco-IOS-XR-l2vpn-oper
LMP Oper Data	Cisco-IOS-XR-lmp-oper
Explicit Path Oper Data	Cisco-IOS-XR-ip-iep-oper
MPLS-TE Oper Data	Cisco-IOS-XR-mpls-te-oper
Route Policy Oper Data	Cisco-IOS-XR-policy-repository-oper
OSPF Oper Data	Cisco-IOS-XR-ipv4-ospf-oper
RSVP Oper Data	Cisco-IOS-XR-ip-rsvp-oper
LDP Config	Cisco-IOS-XR-mpls-ldp-cfg
LDP Oper Data	Cisco-IOS-XR-mpls-ldp-oper
Call Home Oper Data	Cisco-IOS-XR-call-home-oper
Alarms Oper Data	Cisco-IOS-XR-alarmgr-server-oper
Inventory Oper Data	Cisco-IOS-XR-invmgr-oper
BFD Config	Cisco-IOS-XR-ip-bfd-cfg
BFD Oper Data	Cisco-IOS-XR-ip-bfd-oper
SSH Config	Cisco-IOS-XR-crypto-ssh-cfg
SSH Oper Data	Cisco-IOS-XR-crypto-ssh-oper
Ping	Cisco-IOS-XR-mpls-ping-act
Traceroute	Cisco-IOS-XR-mpls-traceroute-act
CFM	Cisco-IOS-XR-ethernet-cfm-act
CFM Oper Data	Cisco-IOS-XR-ethernet-cfm-oper
OBFL Data Clear	Cisco-IOS-XR-sysadmin-obfl-clear **Note** The loaded modules have no trees to display. This is because they are meant to support other modules such as augment, extend, and so on, and are not intended to be directly consumed by the user.

Functionality	YANG Models
Slice Control	Cisco-IOS-XR-sysadmin-controllers-ncs4k

The following Open Config model is supported:

Functionality	YANG Models
Link Aggregation Control Protocol (LACP)	openconfig-lacp **Note** The minimum value that you can enter for the system priority leaf is "1", although the Yang UI displays "0" as the minimum value.