Programmability Configuration Guide for Cisco NCS 6000 Series Routers, IOS XR Release 7.4.x

First Published: 2021-07-01

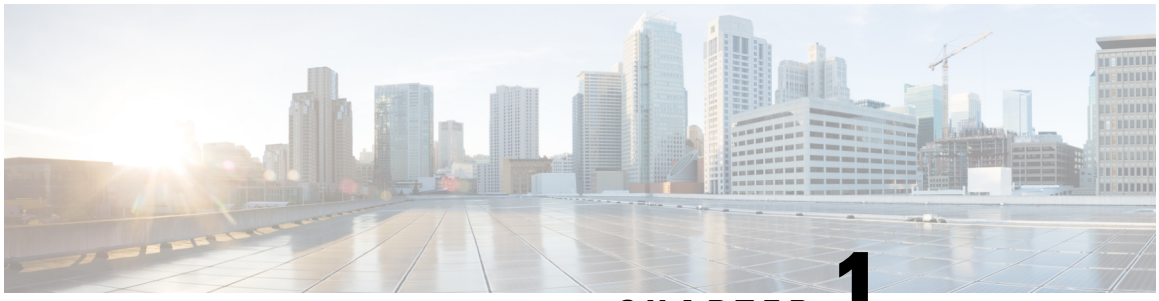
Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

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

Cisco Bug Search Tool

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

CONTENTS



CHAPTER 1

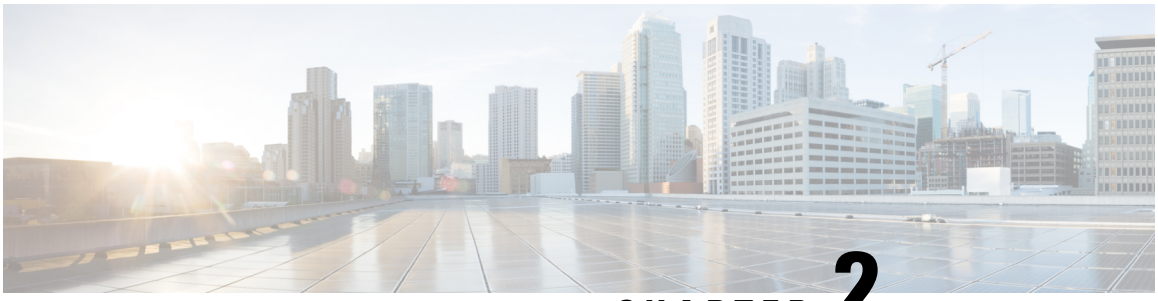
New and Changed Feature Information

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

New and Changed Programmability Features

Feature	Description	Changed in Release	Where Documented
Transitioning Native Models to Unified Models (UM)	Unified models are CLI-based YANG models that are designed to replace the native schema-based models. UM models are generated directly from the IOS XR CLIs and mirror them in several ways. This results in improved usability and faster adoption of YANG models. You can access the new unified models from the Github repository.	Release 7.4.1	Unified Configuration Models, on page 51
CLI to YANG Mapping tool	This tool provides a quick reference for IOS XR CLIs and a corresponding YANG data model that could be used. New command introduced for this feature: **yang describe**	Release 7.4.1	CLI to Yang Mapping Tool, on page 9

Feature	Description	Changed in Release	Where Documented
Unique Commit ID for Configuration State	The network orchestrator is a central point of management for the network and typical workflow involves synchronizing the configuration states of the routers it manages. Loading configurations for comparing the states involves unnecessary data and subsequent comparisons are load intensive. This feature synchronizes the configuration states between the orchestrator and the router using a unique commit ID that the router maintains for each configuration commit. The orchestrator retrieves this commit ID from the router using NETCONF Remote Procedure Calls (RPCs) to identify whether the router has the latest configuration.	Release 7.4.1	Retrieve Transaction ID for NSO Operations, on page 27



CHAPTER 2

Drive Network Automation Using Programmable YANG Data Models

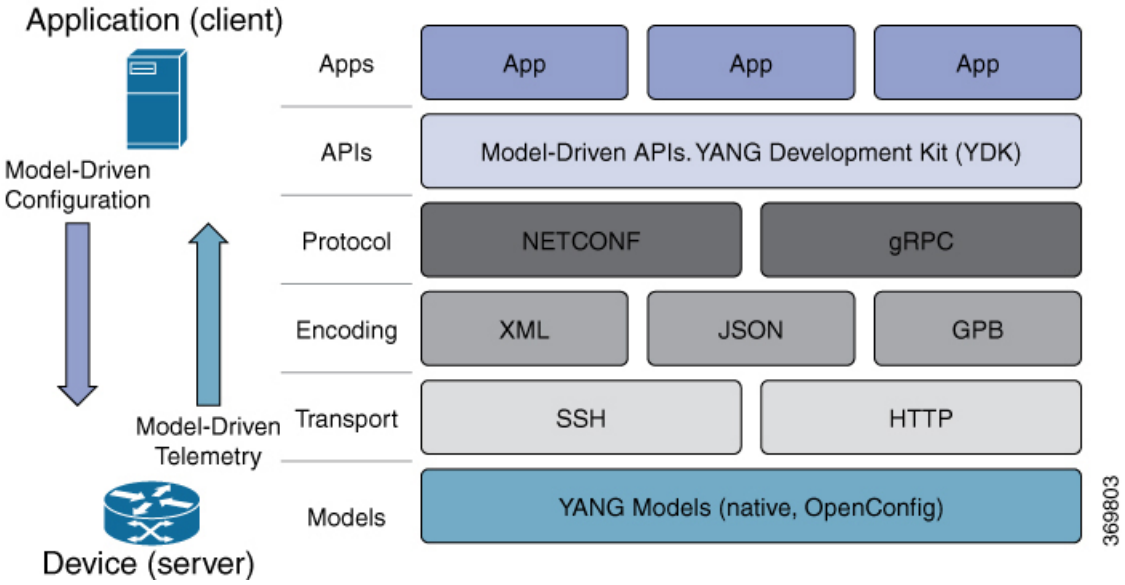
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the 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 (NETCONF Protocol) or google-defined Remote Procedure Calls (gRPC 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

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.

Note gRPC is supported only in 64-bit platforms.

YANG models can be:

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

All data models are stamped with semantic version `1.0.0` as baseline from release 7.0.1 and later.

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.

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

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:

The structure of a data model can be explored using a YANG validator tool such as pyang and the data model can be formatted in a tree structure. The following example shows the CDP operational model in tree format.

```
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

.......................... (snipped) ...........................
```

Access the Data Models

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

CLI-based YANG data models, also known as unified configuration models were introduced in Cisco IOS XR, Release 7.0.1. The new set of unified YANG config models are built in alignment with the CLI commands.

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 type="subtree">
     <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="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>1.0.0</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>1.0.0</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>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
    <location>NETCONF</location>
</schema>

------------<snipped>--------------
<schema>
    <identifier>openconfig-aft-types</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/fib-types</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>openconfig-mpls-ldp</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/ldp</namespace>
    <location>NETCONF</location>
</schema>
</schemas>
</netconf-state>
------------<truncated>--------------
```

CLI to Yang Mapping Tool

Table 1: Feature History Table

Feature Name	Release Information	Description
CLI to YANG Mapping Tool	Release 7.4.1	This tool provides a quick reference for IOS XR CLIs and a corresponding YANG data model that could be used. New command introduced for this feature: **yang describe**

CLI commands are widely used for configuring and extracting the operational details of a router. But 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 router using Yang data models. However, owing to the large number of CLI commands, it is cumbersome to determine the mapping between the CLI command and its associated data model.

The CLI to Yang describer tool is a component in the IOS XR software. It helps in mapping the CLI command with its equivalent data models. With this tool, network automation using data models can be adapted with ease.

The tool simulates the CLI command and displays the following data:

- Yang model mapping to the CLI command
- List of the associated sensor paths

To retrieve the Yang equivalent of a CLI, use the following command:

```
Router#yang-describe ?
  configuration  Describe configuration commands(cisco-support)
  operational    Describe operational commands(cisco-support)
```

The tool supports description of both operational and configurational commands.

Example: Configuration Data

In the following example, the Yang paths for configuring the MPLS label range with minimum and maximum static values are displayed:

```
Router#yang-describe configuration mpls label range table 0 34000 749999 static 34000 99999
Mon May 10 12:37:27.192 UTC
YANG Paths:
  Cisco-IOS-XR-um-mpls-lsd-cfg:mpls/label/range/table-0
  Cisco-IOS-XR-mpls-lsd-cfg:mpls-lsd/label-databases/label-database/label-range
  Cisco-IOS-XR-mpls-lsd-cfg:mpls-lsd/label-databases/label-database/label-range/minvalue
  Cisco-IOS-XR-mpls-lsd-cfg:mpls-lsd/label-databases/label-database/label-range/max-value

Cisco-IOS-XR-mpls-lsd-cfg:mpls-lsd/label-databases/label-database/label-range/min-static-value


Cisco-IOS-XR-mpls-lsd-cfg:mpls-lsd/label-databases/label-database/label-range/max-static-value
```

In the following example, the Yang paths for configuring the gRPC address are displayed:

```
Router#yang-describe configuration grpc address-family ipv4
Mon May 10 12:39:56.652 UTC
YANG Paths:
  Cisco-IOS-XR-man-ems-cfg:grpc/enable
  Cisco-IOS-XR-man-ems-cfg:grpc/address-family
```

Example: Operational Data

The operational data includes support for the `show` CLI commands.

The example shows the Yang paths to retrieve the operational data for MPLS interfaces:

```
Router#yang-describe operational show mpls interfaces
Mon May 10 12:34:05.198 UTC
YANG Paths:
  Cisco-IOS-XR-mpls-lsd-oper:mpls-lsd/interfaces/interface
```

The following example shows the Yang paths to retrieve the operational data for Virtual Router Redundancy Protocol (VRRP):

```
Router#yang-describe operational show vrrp brief
Mon May 10 12:34:38.041 UTC
YANG Paths:
  Cisco-IOS-XR-ipv4-vrrp-oper:vrrp/ipv4/virtual-routers/virtual-router
  Cisco-IOS-XR-ipv4-vrrp-oper:vrrp/ipv6/virtual-routers/virtual-router
```

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)
- RPC framework (gRPC) by Google

Note gRPC is supported only in 64-bit platforms.

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 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 Use NETCONF Protocol to Define Network Operations with Data Models, on page 15.

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. You 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. To get started with issuing NETCONF RPCs to configure network features using data models, see Use gRPC Protocol to Define Network Operations with Data Models, on page 35.

Note gRPC is supported only in 64-bit platforms.

YANG Actions

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

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

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
clear isis	Cisco-IOS-XR-isis-act
clear bgp	Cisco-IOS-XR-ipv4-bgp-act

Example: PING NETCONF Action

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

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

```
  <destination>
   <destination>1.2.3.4</destination>
  </destination>
 </ping>
</rpc>
```

This section shows the NETCONF action response from the router.

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

Example: XR Process Restart Action

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

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

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

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

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:

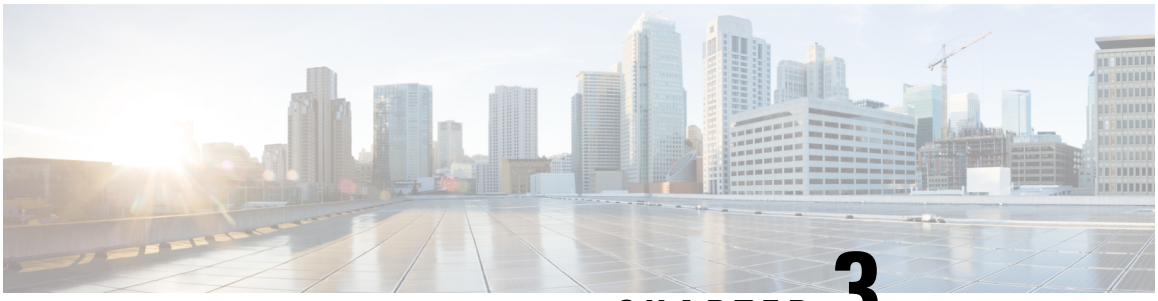
RPC request:

```
<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
```



CHAPTER 3

Use NETCONF Protocol to Define Network Operations with Data Models

Table 2: Feature History Table

Feature Name	Release Information	Description
Unified NETCONF V1.0 and V1.1	Release 7.3.1	Cisco IOS XR supports NETCONF 1.0 and 1.1 programmable management interfaces. With this release, a client can choose to establish a NETCONF 1.0 or 1.1 session using a separate interface for both these formats. This enhancement provides a secure channel to operate the network with both interface specifications.

XR devices ship with the YANG files that define the data models they support. Using a management protocol such as NETCONF or gRPC, 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, use the **ssh server capability netconf-xml** command to reach XML subsystem on port 22.

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. At the end of each message, the NETCONF agent sends the `]]>]]>` marker. Sessions are terminated using `close` or `kill` messages.

Configure NETCONF Agent

To configure a NETCONF TTY agent, use the **netconf agent tty** command. In this example, you configure the *throttle* and *session timeout* parameters:

```
netconf agent tty
         throttle (memory | process-rate)
         session timeout
```

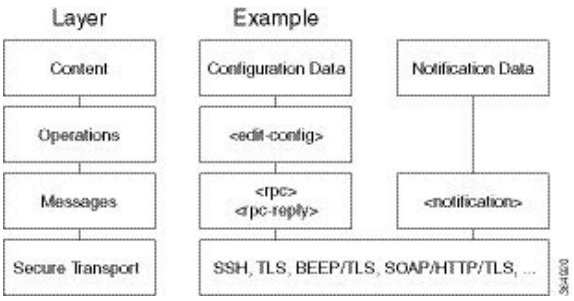
To enable the NETCONF SSH agent, use the following command:

```
ssh server v2
netconf agent tty
```

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 17
- Retrieve Default Parameters Using with-defaults Capability, on page 21
- Retrieve Transaction ID for NSO Operations, on page 27
- Set Router Clock Using Data Model in a NETCONF Session, on page 29

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
```

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	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>`

NETCONF Operation	Description	Example
<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>
<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>

NETCONF Operation	Description	Example

<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>`

<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>`

NETCONF Operation	Description	Example
<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>

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`.

Retrieve Default Parameters Using with-defaults Capability

You can retrieve the default parameters of a data node using a NETCONF protocol operation that includes the `<with-default>` capability.

NETCONF servers report default data nodes in response to RPC requests in the following ways:

- report-all: All data nodes are reported
- trim: Data nodes set to the YANG default aren't reported
- explicit: Data nodes set to the YANG default by the client are reported

Cisco IOS XR routers support only the explicit basic mode. A server that uses this mode must consider any data node that isn't explicitly set to be the default data. As per RFC 6243, the routers support `<with-defaults>` capability for configuration and state data.

The `<with-defaults>` capability indicates which default-handling basic mode is supported by the server. It also indicates support for additional defaults retrieval modes. These retrieval modes allow a NETCONF client to control whether the server returns the default data.

By default, `<with-defaults>` capability is disabled. To enable this capability, use **netconf-yang agent with-defaults** command in Config mode. Once enabled, the capability is applied to all netconf-yang requests.

Note Currently, the `<with-defaults>` capability is supported only for `openconfig-interface.yang` data model.

The `<get>`, `<get-config>`, `<copy-config>` and ,`<edit-config>` operations support `with-defaults` capability.

Example 1: Create Operation

A valid `create` operation attribute for a data node that is set by the server to its schema default value must succeed. It is set or used by the device whenever the NETCONF client does not provide a specific value for the relevant data node. In the following example, an `edit-config` request is sent to create a configuration:

<edit-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:43efc290-c312-4df0-bb1b-a6e0bf8aac50">
<edit-config>
<target>
<candidate/>
</target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface>
<index>2</index>
<config>
<enabled xc:operation="create">false</enabled>
<index xc:operation="create">2</index>
</config>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
```

```
</config>
</edit-config>
</rpc>
```

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>
```

Commit the configuration.

```
[host 172.x.x.x session-id 2985924161] Requesting 'Commit'
[host 172.x.x.x session-id 2985924161] Sending:
<?xml version="1.0" encoding="UTF-8"?><nc:rpc
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:295eff87-1fb6-4f84-bb7d-c40b268eab1b"><nc:commit/></nc:rpc>

[host 172.x.x.x session-id 2985924161] Received:
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:295eff87-1fb6-4f84-bb7d-c40b268eab1b"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
CREATE operation completed
```

A `create` operation attribute for a data node that has been set by a client to its schema default value must fail with a `data-exists` error tag. The client can only create a default node that was not previously created by it. Else, the operation is rejected with the `data-exists` message.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1f29267f-7593-4a3c-8382-6ab9bec323ca">
<edit-config>
 <target>
  <candidate/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
    <name>TenGigE0/0/0/0</name>
     <subinterfaces>
      <subinterface>
       <index>2</index>
        <config>
         <enabled xc:operation="create">false</enabled>
         <index xc:operation="create">2</index>
        </config>
      </subinterface>
     </subinterfaces>
    </interface>
   </interfaces>
 </config>
</edit-config>
</rpc>

[host 172.x.x.x session-id 2985924161] Received:
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:1f29267f-7593-4a3c-8382-6ab9bec323ca"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
```

```
   <error-tag>data-exists</error-tag>
   <error-severity>error</error-severity>
   <error-path
xmlns:ns1="http://openconfig.net/yang/interfaces">ns1:interfaces/ns1:interface[name =
'TenGigE0/0/0/0']/ns1:subinterfaces/ns1:subinterface[index = '2']/ns1:config</error-path>
 </rpc-error>
</rpc-reply>
```

Example 2: Delete Operation

A valid `delete` operation attribute for a data node set by a client to its schema default value must succeed. Whereas a valid `delete` operation attribute for a data node set by the server to its schema default value fails with a `data-missing` error tag.

<edit-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:de95a248-29d7-4030-8351-cef8b8d47cdb">
<edit-config>
<target>
<candidate/>
</target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface xc:operation="delete">
<index>2</index>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</config>
</edit-config>
</rpc>
```

Response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:de95a248-29d7-4030-8351-cef8b8d47cdb"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error>
<error-type>application</error-type>
<error-tag>data-missing</error-tag>
<error-severity>error</error-severity>
<error-path xmlns:ns1="http://openconfig.net/yang/interfaces">ns1:interfaces/ns1:
interface[name = 'TenGigE0/0/0/0']/ns1:subinterfaces/ns1:subinterface[index =
'2']/ns1:config</error-path></rpc-error>
</rpc-reply>
```

Example 3: Copy Configuration

In the following example, a `copy-config` request is sent to copy a configuration.

<copy-config> request sent to the NETCONF agent:

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

```
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
    <name>TenGigE0/0/0/0</name>
    <subinterfaces>
     <subinterface>
      <index>2</index>
      <config>
       <index>2</index>
      </config>
     </subinterface>
    </subinterfaces>
   </interface>

  </interfaces>
</config>
</source>
<with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
</copy-config>
</rpc>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
  <commit/>
</rpc>
```

The show run command shows the copied configuration.

```
Router#show run
<data and time stamp>
Building configuration...
!! IOS XR Configuration 7.2.1
!! Last configuration change at <data and time stamp> by root
!
interface TenGigE0/0/0/0.2
!
end
```

Example 4: Get Configuration

The following example shows a `get-config` request with `explicit` mode to query the default parameters from the oc-interfaces.yang data model. The client gets the configuration values of what it sets.

<get-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:63a49626-9f90-4ebe-89fd-741410cddf29">
 <get-config>
 <source>
 <running/>
 </source>
 <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
 <filter type="subtree">
 <interfaces xmlns="http://openconfig.net/yang/interfaces"/>
 </filter>
 </get-config>
 </rpc>
```

<get-config> response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:99d8b2d0-ab05-474a-bc02-9242ba511308"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
```

```
 <interfaces xmlns="http://openconfig.net/yang/interfaces">
  <interface>
    <name>TenGigE0/0/0/0</name>
      <subinterfaces>
      <subinterface>
        <index>2</index>
          <config>
           <index>2</index>
           <enabled>false</enabled>
          </config>
        <ipv6 xmlns="http://openconfig.net/yang/interfaces/ip">
        <config>
           <enabled>false</enabled>
          </config>
         </ipv6>
       </subinterface>
    </subinterfaces>
   </interface>
 <interface>
  <name>MgmtEth0/RSP0/CPU0/0</name>
   <config>
    <name>MgmtEth0/RSP0/CPU0/0</name>
     <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:ethernetCsmacd</type>

   </config>
    <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
     <config>
      <auto-negotiate>false</auto-negotiate>
     </config>
    </ethernet>
  <subinterfaces>
 <subinterface>
 <index>0</index>
<ipv4 xmlns="http://openconfig.net/yang/interfaces/ip">
<addresses>
 <address>
  <ip>172.xx.xx.xx</ip>
   <config>
    <ip>172.xx.xx.xx</ip>
    <prefix-length>24</prefix-length>
   </config>
 </address>
</addresses>
</ipv4>
</subinterface>
 </subinterfaces>
  </interface>
   <interface>
    <name>MgmtEth0/RSP1/CPU0/0</name>
    <config>
     <name>MgmtEth0/RSP1/CPU0/0</name>
     <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:ethernetCsmacd</type>
     <enabled>false</enabled>
  </config>
   <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
   <config>
     <auto-negotiate>false</auto-negotiate>
  </config>
 </ethernet>
</interface>
</interfaces>
</data>
</rpc-reply>
READ operation completed
```

Example 5: Get Operation

The following example shows a `get` request with `explicit` mode to query the default parameters from the oc-interfaces.yang data model.

<get-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:d8e52f0f-ceac-4193-89f6-d377ab8292d5">
<get>
<with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
<filter type="subtree">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface>
<index>2</index>
<state/>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</filter>
</get>
</rpc>
```

<get> response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:933df011-191f-4f31-9549-c4f7f6edd291"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
   <interfaces xmlns="http://openconfig.net/yang/interfaces">
    <interface>
    <name>TenGigE0/0/0/0</name>
     <subinterfaces>
      <subinterface>
      <index>2</index>
     <state>
      <index>2</index>
      <name>TenGigE0/0/0/0.2</name>
      <enabled>false</enabled>
      <admin-status>DOWN</admin-status>
      <oper-status>DOWN</oper-status>
      <last-change>0</last-change>
      <counters>
      <in-unicast-pkts>0</in-unicast-pkts>
      <in-pkts>0</in-pkts>
      <in-broadcast-pkts>0</in-broadcast-pkts>
      <in-multicast-pkts>0</in-multicast-pkts>
      <in-octets>0</in-octets>
      <out-unicast-pkts>0</out-unicast-pkts>
      <out-broadcast-pkts>0</out-broadcast-pkts>
      <out-multicast-pkts>0</out-multicast-pkts>
      <out-pkts>0</out-pkts>
      <out-octets>0</out-octets>
      <out-discards>0</out-discards>
      <in-discards>0</in-discards>
      <in-unknown-protos>0</in-unknown-protos>
      <in-errors>0</in-errors>
      <in-fcs-errors>0</in-fcs-errors>
```

```
      <out-errors>0</out-errors>
     <carrier-transitions>0</carrier-transitions>
    <last-clear>2020-03-02T15:35:30.927+00:00</last-clear>
   </counters>
  <ifindex>92</ifindex>
   <logical>true</logical>
  </state>
 </subinterface>
 </subinterfaces>
 </interface>
 </interfaces>
 </data>
</rpc-reply>
READ operation completed
```

Retrieve Transaction ID for NSO Operations

Table 3: Feature History Table

Feature Name	Release Information	Description
Unique Commit ID for Configuration State	Release 7.4.1	The network orchestrator is a central point of management for the network and typical workflow involves synchronizing the configuration states of the routers it manages. Loading configurations for comparing the states involves unnecessary data and subsequent comparisons are load intensive. This feature synchronizes the configuration states between the orchestrator and the router using a unique commit ID that the router maintains for each configuration commit. The orchestrator retrieves this commit ID from the router using NETCONF Remote Procedure Calls (RPCs) to identify whether the router has the latest configuration.

Cisco Network Services Orchestrator (NSO) is a data model-driven platform for automating your network orchestration. NSO uses NETCONF-based Network Element Drivers (NED) to synchronize the configuration states of the routers it manages. NEDs comprise of the network-facing part of NSO and communicate over the native protocol supported by the router, such as Network Configuration Protocol (NETCONF).

IOS XR configuration manager maintains commit IDs (also known as the transaction IDs) for each commit operation. The manageability interfaces use these IDs. Currently, the operational data model provides a list of up to 100 last commits for NETCONF requests. The YANG client querying the last commit ID collects the entire list and finds the latest ID. Loading configurations for comparison to the orchestrator's configuration state can involve huge redundant data. The subsequent comparisons are also load intensive.

To overcome these limitations, the router maintains a unique last commit ID that is ideal for NSO operations. This ID indicates the latest configuration state on the router. The ID provides a one-step operation and increases the performance of configuration updates for the orchestrator.

An augmented configuration manageability model `Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper` provides a single `last-commit-id` for the unique commit state. This model is available as part of the base package.

The following table lists the synchronization support between NSO and the IOS XR variants:

Entity	32-bit Routers	64-bit Routers (Releases Earlier than 7.4.1)	64-bit Routers (Releases 7.4.1 and Later)
cfgmgr	Yes	Yes	Yes
sysadmin	No	Yes	Yes
cfgmgr-aug	No	No	Yes
Leaf Data	commit-id	NA	cfgmgr-aug
Check synchronization (NSO functionality from release 7.4.1 and later)	Yes	No	Yes

Where:

- commit-id represents `Cisco-IOS-XR-config-cfgmgr-exec-oper:config-manager/global/config-commit/commits/commit/commit-id`
- cfgmgr is the XR configuration manager
- sysadmin represents the `Cisco-IOS-XR-sysadmin-system` data model
- cfgmgr-aug represents the `Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper` data model

The last commit ID is obtained from the configuration manager. The following example shows a sample NETCONF request and response to retrieve the commit ID:

```
Request:
<rpc message-id="test" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get>
  <filter type="subtree">
    <config-manager xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-oper">
      <global>
        <config-commit>
          <last-commit-id
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper"/>
        </config-commit>
      </global>
    </config-manager>
 </filter>
</get>
</rpc>


Response:
<rpc-reply message-id="test" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
```

```
  <config-manager xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-oper">
   <global>
    <config-commit>
     <last-commit-id
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper">
         XR:1000000009;Admin:1595-891537-949905</last-commit-id>
    </config-commit>
   </global>
  </config-manager>
 </data>
</rpc-reply>
```

Set Router Clock Using Data Model in a NETCONF Session

NETCONF Protocol 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.

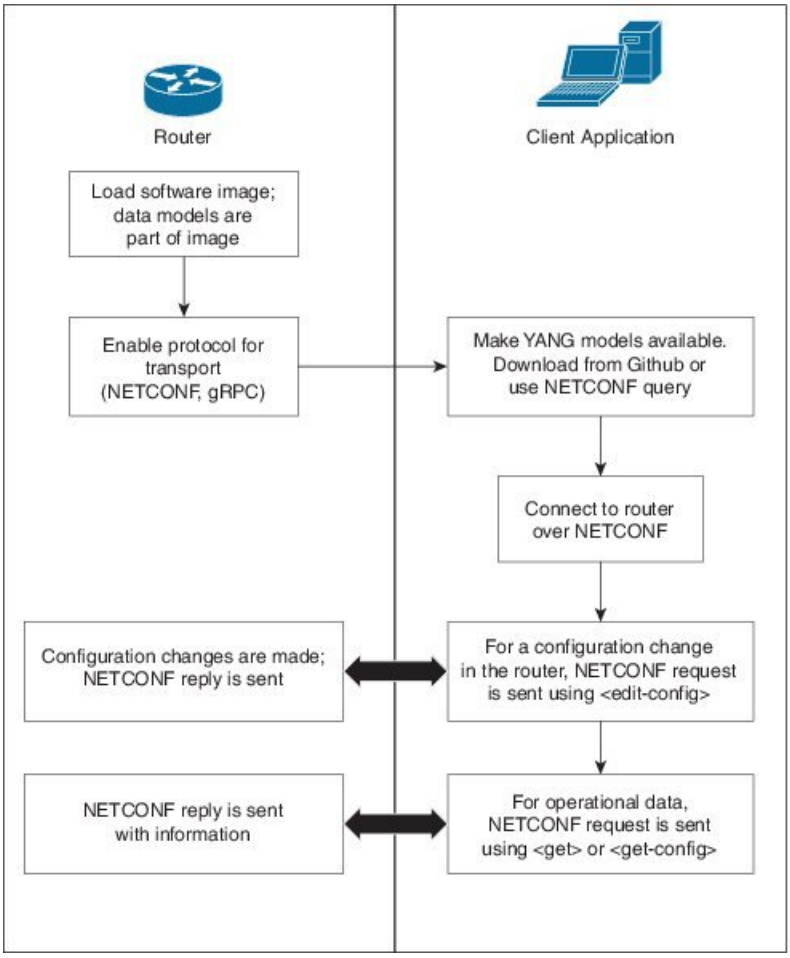
The process for using data models involves:

- 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.

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 6000 Series Routers*.

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

Figure 3: Process for Using Data Models

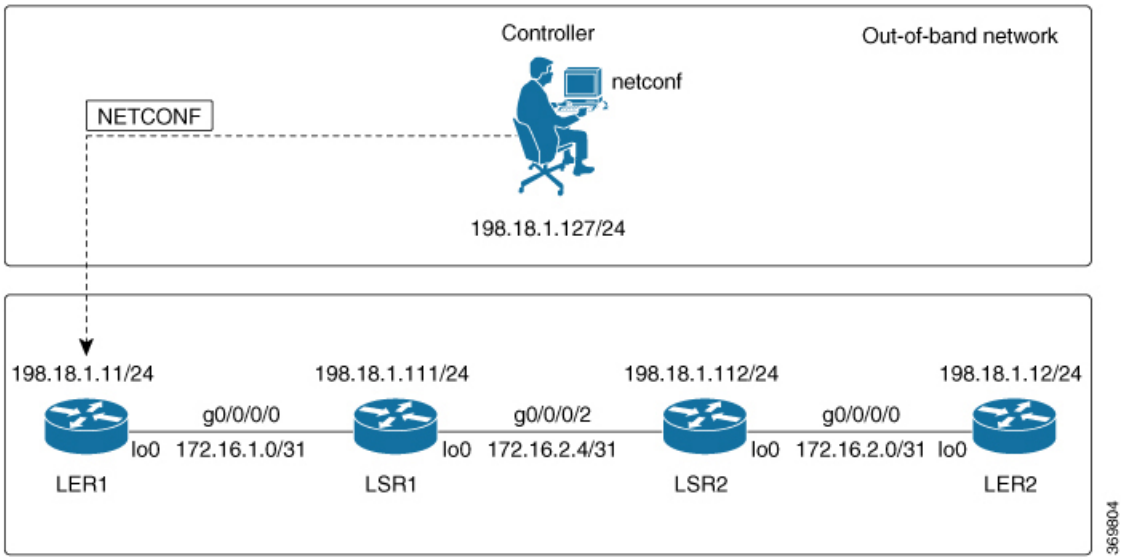


In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

Consider a network topology with four routers and one controller. The network consists of label edge routers (LER) and label switching routers (LSR). Two routers LER1 and LER2 are label edge routers, and two routers LSR1 and LSR2 are label switching routers. A host is the controller with a gRPC client. The controller communicates with all routers through an out-of-band network. All routers except LER1 are pre-configured with proper IP addressing and routing behavior. Interfaces between routers have a point-to-point configuration with `/31` addressing. Loopback prefixes use the format `172.16.255.x/32`.

The following image illustrates the network topology:

Figure 4: Network Topology for gRPC session



You use Cisco IOS XR native models `Cisco-IOS-XR-infra-clock-linux-cfg.yang` and `Cisco-IOX-XR-shellutil-oper` to programmatically configure the router clock. You can explore the structure of the data model using YANG validator tools such as pyang.

Before you begin

Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information, see Access the Data Models, on page 7.

Configure Router Clock

Step 1 Explore the native configuration model for the system local time zone.

Example:

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-infra-infra-clock-linux-cfg.yang
module: Cisco-IOS-XR-infra-infra-clock-linux-cfg
    +--rw clock
        +--rw time-zone!
        +--rw time-zone-name string
        +--rw area-name string
```

Step 2 Explore the native operational state model for the system time.

Example:

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-shellutil-oper.yang
module: Cisco-IOS-XR-shellutil-oper
    +--ro system-time
        +--ro clock
        | +--ro year? uint16
        | +--ro month? uint8
        | +--ro day? uint8
```

```
        | +--ro hour? uint8
        | +--ro minute? uint8
        | +--ro second? uint8
        | +--ro millisecond? uint16
        | +--ro wday? uint16
        | +--ro time-zone? string
        | +--ro time-source? Time-source
        +--ro uptime
            +--ro host-name? string
            +--ro uptime? uint32
```

Step 3 Retrieve the current time on router LER1.

Example:

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ netconf get --filter xr-system-time-oper.xml
198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>17</hour>
        <minute>30</minute>
        <second>37</second>
        <millisecond>690</millisecond>
        <wday>1</wday>
        <time-zone>UTC</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>851237</uptime>
    </uptime>
</system-time>
```

Notice that the timezone UTC indicates that a local timezone is not set.

Step 4 Configure Pacific Standard Time (PST) as local time zone on LER1.

Example:

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
  <system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
```

```
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

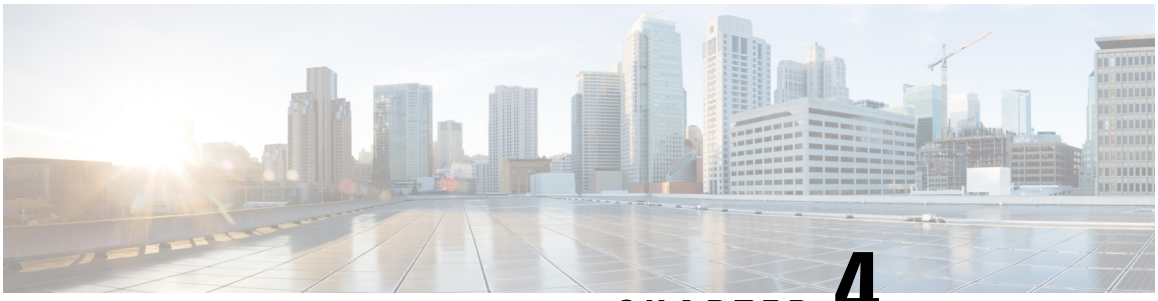
View the Router Clock

Verify that the router clock is set to PST time zone.

```
controller:netconf$ more xr-system-time-oper.xml
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>

controller:netconf$ netconf get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2018</year>
        <month>12</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

In summary, router LER1, which had no local timezone configuration, is programmatically configured using data models.



CHAPTER 4

Use gRPC 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 or gRPC, you can programmatically query a device for the list of models it supports and retrieve the model files.

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. You define the structure using 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.

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.

Note All 64-bit IOS XR platforms support gRPC and TCP protocols. All 32-bit IOS XR platforms support only TCP protocol.

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) {};

}

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

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

This article describes, with a use case to configure interfaces on a router, how data models helps in a faster programmatic and standards-based configuration of a network, as comapared to CLI.

gRPC Operations

You can issue the following gRPC operations:

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

gRPC Operation to Get Configuration

This example shows how a gRPC GetConfig 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.

gRPC Request (Client to Router)	gRPC Response (Router to Client)

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

The subscription in a gNMI does not require prior sensor path configuration on the target device. Sensor paths are requested by the collector (such as pipeline), and the subscription mode can be specified for each path.

gNMI 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

gNMI defines 3 modes for a streaming subscription that indicates how the router must return data in a subscription:

- A `SAMPLE` mode is cadence-based subscription supported for all the operational models.
- An `ON_CHANGE` mode is event-based subscription. In this mode, only the state leaf supports on_change events.
- A `TARGET_DEFINED` mode allows the target to determine the best type of subscription to be created on a per-leaf basis.

When a client creates a subscription specifying the `TARGET_DEFINED` mode, the target, here the router, determine the best type of subscription to be created on a per-leaf basis. If the path specified within the message refers to some leaves which are event-driven, then an `ON_CHANGE` subscription is created. Else, a `SAMPLE` subscription is created.

Note In Cisco IOS XR Release 6.3.1, the `TARGET_DEFINED` subscription mode is supported only for sensor paths of OpenConfig model; native model is not supported. The supported models are: OC Interfaces, OC Telemetry, OC Shell Util, OC System NTP and OC Platform.

For more information about gNMI, see Github.

gRPC Network Operations Interface

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices. Extensible Manageability Services (EMS) gNOI is the Cisco IOS XR implementation of gNOI.

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

GNOI supports the following remote procedure calls (RPCs):

- System:
  - Reboot
  - RebootStatus
  - SetPackage
  - SetPackageRemote
  - Ping
  - Traceroute
  - Time
  - SwitchControlProcessor
- File
  - Get
  - Remove
  - Stat
  - Put
  - TransferToRemote
- Cert
  - Rotate
  - Install
  - GetCertificates
  - RevokeCertificates
  - CanGenerateCSR
- Interface
  - SetLoopbackMode
  - GetLoopbackMode

    - ClearInterfaceCounters
- Layer2
    - ClearLLDPInterface
- BGP
    - ClearBGPNeighbor
- BERT
    - StartBERT
    - StopBERT
    - GetBERTResult

The following examples show the representation of few gNOI RPCs:

Get RPC

Streams the contents of a file from the target.

```
RPC to 10.105.57.106:57900
RPC start time: 20:58:27.513638
---------------------File Get Request----------------------
RPC start time: 20:58:27.513668
remote_file: "harddisk:/giso_image_repo/test.log"

---------------------File Get Response----------------------
RPC end time: 20:58:27.518413
contents: "GNOI \n\n"

hash {
method: MD5
hash: "D\002\375h\237\322\024\341\370\3619k\310\333\016\343"
}
```

Remove RPC

Remove the specified file from the target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:07:57.089554
---------------------File Remove Request----------------------
remote_file: "harddisk:/sample.txt"

---------------------File Remove Response----------------------
RPC end time: 21:09:27.796217
File removal harddisk:/sample.txt successful
```

Reboot RPC

Reloads a requested target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:12:49.811536
---------------------Reboot Request----------------------
RPC start time: 21:12:49.811561
```

```
method: COLD
message: "Test Reboot"
subcomponents {
origin: "openconfig-platform"
elem {
name: "components"
}
elem {
name: "component"
key {
key: "name"
value: "0/RP0"
}
}
elem {
name: "state"
}
elem {
name: "location"
}
}
---------------------Reboot Request---------------------
RPC end time: 21:12:50.023604
```

Set Package RPC

Places software package on the target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:12:49.811536
---------------------Set Package Request---------------------
RPC start time: 15:33:34.378745
Sending SetPackage RPC
package {
filename: "harddisk:/giso_image_repo/<platform-version>-giso.iso"
activate: true
}
method: MD5
hash: "C\314\207\354\217\270=\021\341y\355\240\274\003\034\334"
RPC end time: 15:47:00.928361
```

Reboot Status RPC

Returns the status of reboot for the target.

```
RPC to 10.105.57.106:57900
RPC start time: 22:27:34.209473
---------------------Reboot Status Request---------------------
subcomponents {
origin: "openconfig-platform"
elem {
name: "components"
}
elem {
name: "component"
key {
key: "name"
value: "0/RP0"
}
}
elem {
name: "state"
}
elem
```

```
name: "location"
}
}

RPC end time: 22:27:34.319618

--------------------Reboot Status Response--------------------
Active : False
Wait : 0
When : 0
Reason : Test Reboot
Count : 0
```

To send gNOI RPC requests, user needs a client that implements the gNOI client interface for each RPC.

All messages within the gRPC service definition are defined as protocol buffers (proto files). gNOI OpenConfig proto files are located in Github repository.

Configure Interfaces Using Data Models in a gRPC Session

Google-defined remote procedure call (gRPC Protocol) is an open-source RPC framework. gRPC supports IPv4 and IPv6 address families. 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 gRPC communication protocol.
- Manage the configuration of the router from the client using data models.

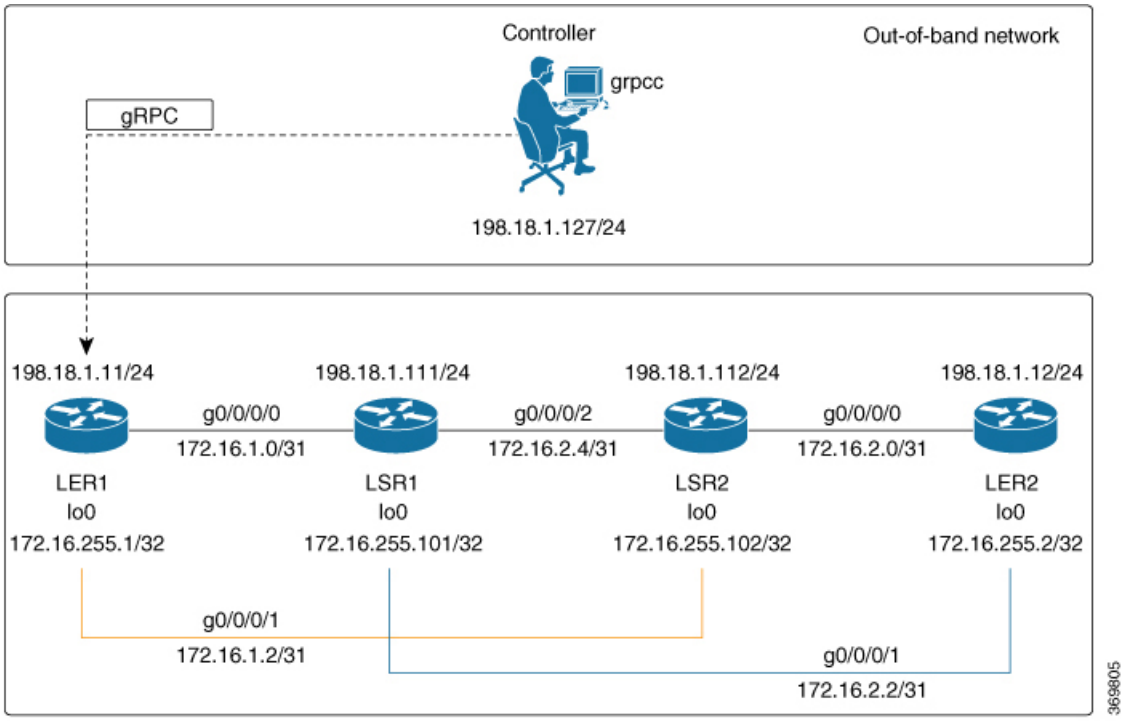
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network 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 6000 Series Routers*.

In this section, you use native data models to configure loopback and ethernet interfaces on a router using a gRPC session.

Consider a network topology with four routers and one controller. The network consists of label edge routers (LER) and label switching routers (LSR). Two routers LER1 and LER2 are label edge routers, and two routers LSR1 and LSR2 are label switching routers. A host is the controller with a gRPC client. The controller communicates with all routers through an out-of-band network. All routers except LER1 are pre-configured with proper IP addressing and routing behavior. Interfaces between routers have a point-to-point configuration with `/31` addressing. Loopback prefixes use the format `172.16.255.x/32`.

The following image illustrates the network topology:

Figure 5: Network Topology for gRPC session



You use Cisco IOS XR native model `Cisco-IOS-XR-ifmgr-cfg.yang` to programmatically configure router LER1.

Before you begin

- Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information, see Access the Data Models, on page 7.
- Configure Transport Layer Security (TLS). Enabling gRPC protocol uses the default HTTP/2 transport with no TLS. 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.

Enable gRPC Protocol

To configure network devices and view operational data, gRPC proptocol must be enabled on the server. In this example, you enable gRPC protocol on LER1, the server.

Note Cisco IOS XR 64-bit platforms support gRPC protocol. The 32-bit platforms do not support gRPC protocol.

Step 1 Enable gRPC over an HTTP/2 connection.

Example:

```
Router#configure
Router(config)#grpc
Router(config-grpc)#port <port-number>
```

The port number ranges from 57344 to 57999. If a port number is unavailable, an error is displayed.

Step 2 Set the session parameters.

Example:

```
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

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

Configure Interfaces

In this example, you configure interfaces using Cisco IOS XR native model `Cisco-IOS-XR-ifmgr-cfg.yang`. You gain an understanding about the various gRPC operations while you configure the interface. For the complete list of operations, see gRPC Operations, on page 38. In this example, you merge configurations with `merge-config` RPC, retreive operational statistics using `get-oper` RPC, and delete a configuration using `delete-config` RPC. You can explore the structure of the data model using YANG validator tools such as pyang.

LER1 is the gRPC server, and a command line utility `grpcc` is used as a client on the controller. This utility does not support YANG and, therefore, does not validate the data model. The server, LER1, validates the data mode.

Note The OC interface maps all IP configurations for parent interface under a VLAN with index 0. Hence, do not configure a sub interface with tag 0.

Step 1 Explore the XR configuration model for interfaces and its IPv4 augmentation.

Example:

```
controller:grpc$ pyang --format tree --tree-depth 3 Cisco-IOS-XR-ifmgr-cfg.yang
Cisco-IOS-XR-ipv4-io-cfg.yang
module: Cisco-IOS-XR-ifmgr-cfg
    +--rw global-interface-configuration
    | +--rw link-status? Link-status-enum
    +--rw interface-configurations
        +--rw interface-configuration* [active interface-name]
            +--rw dampening
            | ...
            +--rw mtus
            | ...
            +--rw encapsulation
            | ...
            +--rw shutdown? empty
            +--rw interface-virtual? empty
            +--rw secondary-admin-state? Secondary-admin-state-enum
            +--rw interface-mode-non-physical? Interface-mode-enum
            +--rw bandwidth? uint32
            +--rw link-status? empty
            +--rw description? string
            +--rw active Interface-active
            +--rw interface-name xr:Interface-name
            +--rw ipv4-io-cfg:ipv4-network
            | ...
            +--rw ipv4-io-cfg:ipv4-network-forwarding ...
```

Step 2 Configure a loopback0 interface on LER1.

a) Configure loopback interface `Loopback0` and assign an IP address.

Example:

```
controller:grpc$ more xr-interfaces-lo0-cfg.json
{
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations":
  { "interface-configuration": [
    {
      "active": "act",
      "interface-name": "Loopback0",
      "description": "LOCAL TERMINATION ADDRESS",
      "interface-virtual": [
        null
       ],
       "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
        "addresses": {
            "primary": {
              "address": "172.16.255.1",
              "netmask": "255.255.255.255"
       }
      }
     }
    }
```

```
    ]
  }
}
```

b) Merge the configuration.

Example:

```
controller:grpc$ grpcc -username admin -password admin -oper merge-config
-server_addr 198.18.1.11:57400 -json_in_file xr-interfaces-lo0-cfg.json
emsMergeConfig: Sending ReqId 1
emsMergeConfig: Received ReqId 1, Response '
'
```

Step 3 Configure the ethernet interface on LER1.

a) Configure interface `GigabitEthernet0/0/0/0` on LER1.

Example:

```
controller:grpc$ more xr-interfaces-gi0-cfg.json
{
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
   "interface-configuration": [
    {
     "active": "act",
     "interface-name": "GigabitEthernet0/0/0/0",
     "description": "CONNECTS TO LSR1 (g0/0/0/0)",
     "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
        "addresses": {
         "primary": {
            "address": "172.16.1.0",
            "netmask": "255.255.255.254"
      }
     }
    }
   }
  ]
 }
}
```

b) Merge the configuration.

Example:

```
controller:grpc$ grpcc -username admin -password admin -oper merge-config
-server_addr 198.18.1.11:57400 -json_in_file xr-interfaces-gi0-cfg.json
emsMergeConfig: Sending ReqId 1
emsMergeConfig: Received ReqId 1, Response '
'
```

Step 4 Enable the ethernet interface `GigabitEthernet 0/0/0/0` on LER1 to bring up the interface. To do this, delete `shutdown` configuration for the interface.

Example:

```
controller:grpc$ grpcc -username admin -password admin -oper delete-config
-server_addr 198.18.1.11:57400 -yang_path "$(< xr-interfaces-gi0-shutdown-cfg.json )"
emsDeleteConfig: Sending ReqId 1, yangJson {
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
   "interface-configuration": [
   {
```

```
    "active": "act",
    "interface-name": "GigabitEthernet0/0/0/0",
    "shutdown": [
      null
    ]
   }
  ]
 }
}
emsDeleteConfig: Received ReqId 1, Response ''
```

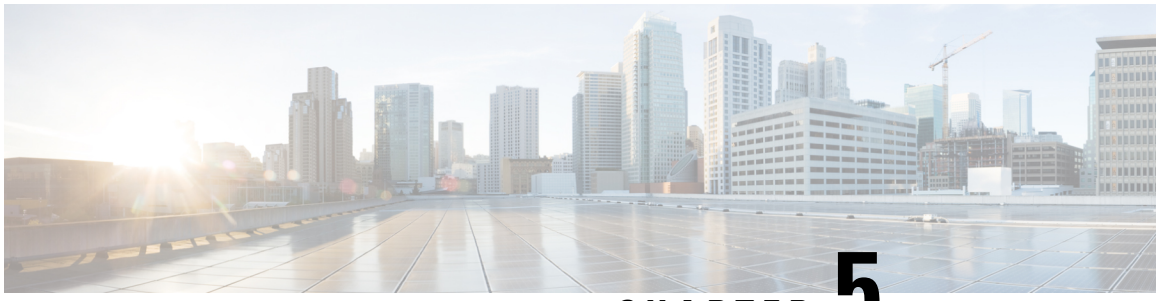
Verify the Interface State

Verify that the loopback interface and the ethernet interface on router LER1 are operational.

```
controller:grpc$ grpcc -username admin -password admin -oper get-oper
-server_addr 198.18.1.11:57400 -oper_yang_path "$(< xr-interfaces-briefs-oper-filter.json
)"
emsGetOper: Sending ReqId 1, yangPath {
  "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
    "interface-briefs": [
      null
     ]
  }
}
{ "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
  "interface-briefs": {
    "interface-brief": [
    {
      "interface-name": "GigabitEthernet0/0/0/0",
      "interface": "GigabitEthernet0/0/0/0",
      "type": "IFT_GETHERNET",
      "state": "im-state-up",
      "actual-state": "im-state-up",
      "line-state": "im-state-up",
      "actual-line-state": "im-state-up",
      "encapsulation": "ether",
      "encapsulation-type-string": "ARPA",
      "mtu": 1514,
      "sub-interface-mtu-overhead": 0,
      "l2-transport": false,
      "bandwidth": 1000000
     },
     {
      "interface-name": "GigabitEthernet0/0/0/1",
      "interface": "GigabitEthernet0/0/0/1",
      "type": "IFT_GETHERNET",
      "state": "im-state-up",
      "actual-state": "im-state-up",
      "line-state": "im-state-up",
      "actual-line-state": "im-state-up",
      "encapsulation": "ether",
      "encapsulation-type-string": "ARPA",
      "mtu": 1514,
      "sub-interface-mtu-overhead": 0,
      "l2-transport": false,
      "bandwidth": 1000000
     },
     {
```

```
        "interface-name": "Loopback0",
        "interface": "Loopback0",
        "type": "IFT_LOOPBACK",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "loopback",
        "encapsulation-type-string": "Loopback",
        "mtu": 1500,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 0
      },
      {
        "interface-name": "MgmtEth0/RP0/CPU0/0",
        "interface": "MgmtEth0/RP0/CPU0/0",
        "type": "IFT_ETHERNET",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "ether",
        "encapsulation-type-string": "ARPA",
        "mtu": 1514,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 1000000
      },
      {
        "interface-name": "Null0",
        "interface": "Null0",
        "type": "IFT_NULL",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "null",
        "encapsulation-type-string": "Null",
        "mtu": 1500,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 0
       }
      ]
     }
    }
}
emsGetOper: ReqId 1, byteRecv: 2325
```


In summary, router LER1, which had minimal configuration, is now programmatically configured using data models with an ethernet interface and is assigned a loopback address. Both these interfaces are operational and ready for network provisioning operations.



CHAPTER 5

Unified Data Models

CLI-based Yang data models, also known as unified configuration models are introduced in Cisco IOS XR Software Release 7.0.1. The unified models provide a full coverage of the router functionality, and serves as a single abstraction for YANG and CLI commands. Unified models are generated from the CLI and replaces the native schema-based models.

The unified models are available in `pkg/yang` location. The presence of `um` in the model name indicates that the model is a unified model. For example, Cisco-IOS-XR-**um**-<feature>-cfg.yang.

The unified models are available in the Github repository. You can also access the models supported on the router using the following command:

```
Router#run
[node]$cd /pkg/yang
[node:pkg/yang]$ls *Cisco-IOS-XR-*netconf*
Cisco-IOS-XR-man-netconf-cfg.yang
Cisco-IOS-XR-man-netconf-oper-sub1.yang
Cisco-IOS-XR-netconf-oper.yang
Cisco-IOS-XR-um-netconf-yang-cfg.yang
[node:pkg/yang]$
```

• Unified Configuration Models, on page 51

Unified Configuration Models

Table 4: Feature History Table

Feature Name	Release Information	Description
Transitioning Native Models to Unified Models (UM)	Release 7.4.1	Unified models are CLI-based YANG models that are designed to replace the native schema-based models. UM models are generated directly from the IOS XR CLIs and mirror them in several ways. This results in improved usability and faster adoption of YANG models. You can access the new unified models from the Github repository.

The following table lists the unified models supported on Cisco IOS XR routers.

Unified Models	Introduced in Release
Cisco-IOS-XR-um-aaa-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-diameter-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-nacm-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-tacacs-server-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-task-user-cfg	Release 7.4.1
Cisco-IOS-XR-um-banner-cfg	Release 7.4.1
Cisco-IOS-XR-um-bfd-sbfd-cfg	Release 7.4.1
Cisco-IOS-XR-um-call-home-cfg	Release 7.4.1
Cisco-IOS-XR-um-cdp-cfg	Release 7.4.1
Cisco-IOS-XR-um-cef-accounting-cfg	Release 7.4.1
Cisco-IOS-XR-um-cfg-mibs-cfg	Release 7.4.1
Cisco-IOS-XR-um-cli-alias-cfg	Release 7.4.1
Cisco-IOS-XR-um-clock-cfg	Release 7.4.1
Cisco-IOS-XR-um-config-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-breakout-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-optics-cfg	Release 7.4.1
Cisco-IOS-XR-um-control-plane-cfg	Release 7.4.1
Cisco-IOS-XR-um-crypto-cfg	Release 7.4.1
Cisco-IOS-XR-um-domain-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-cfm-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-oam-cfg	Release 7.4.1
Cisco-IOS-XR-um-exception-cfg	Release 7.4.1
Cisco-IOS-XR-um-flowspec-cfg	Release 7.4.1
Cisco-IOS-XR-um-frequency-synchronization-cfg	Release 7.4.1
Cisco-IOS-XR-um-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-port-range-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-profile-cfg	Release 7.4.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-ip-virtual-cfg	Release 7.4.1
Cisco-IOS-XR-um-ipsla-cfg	Release 7.4.1
Cisco-IOS-XR-um-l2vpn-cfg	Release 7.4.1
Cisco-IOS-XR-um-line-cfg	Release 7.4.1
Cisco-IOS-XR-um-line-exec-timeout-cfg	Release 7.4.1
Cisco-IOS-XR-um-line-general-cfg	Release 7.4.1
Cisco-IOS-XR-um-line-timestamp-cfg	Release 7.4.1
Cisco-IOS-XR-um-lldp-cfg	Release 7.4.1
Cisco-IOS-XR-um-location-cfg	Release 7.4.1
Cisco-IOS-XR-um-logging-cfg	Release 7.4.1
Cisco-IOS-XR-um-logging-correlator-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-domain-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-dynamic-flows-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-cbqosmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-fabric-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-ifmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-rfmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-sensormib-cfg	Release 7.4.1
Cisco-IOS-XR-um-monitor-session-cfg	Release 7.4.1
Cisco-IOS-XR-um-mpls-oam-cfg	Release 7.4.1
Cisco-IOS-XR-um-ntp-cfg	Release 7.4.1
Cisco-IOS-XR-um-pce-cfg	Release 7.4.1
Cisco-IOS-XR-um-pool-cfg	Release 7.4.1
Cisco-IOS-XR-um-priority-flow-control-cfg	Release 7.4.1
Cisco-IOS-XR-um-rcc-cfg	Release 7.4.1
Cisco-IOS-XR-um-router-hsrp-cfg	Release 7.4.1
Cisco-IOS-XR-um-router-vrrp-cfg	Release 7.4.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-service-timestamps-cfg	Release 7.4.1
Cisco-IOS-XR-um-ssh-cfg	Release 7.4.1
Cisco-IOS-XR-um-tcp-cfg	Release 7.4.1
Cisco-IOS-XR-um-telnet-cfg	Release 7.4.1
Cisco-IOS-XR-um-tpa-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-bridgemib-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-config-copy-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-redundancy-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-state-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-flash-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-fru-ctrl-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-ipsec-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-l2tun-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-otn-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-power-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-selective-vrf-download-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-syslog-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-system-cfg	Release 7.4.1
Cisco-IOS-XR-um-udp-cfg	Release 7.4.1
Cisco-IOS-XR-um-vty-pool-cfg	Release 7.4.1
Cisco-IOS-XR-um-xml-agent-cfg	Release 7.4.1
Cisco-IOS-XR-um-conflict-policy-cfg	Release 7.3.1
Cisco-IOS-XR-um-flow-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-access-group-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv4-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv6-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-service-policy-qos-cfg	Release 7.2.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-ipv4-access-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv6-access-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-l2-ethernet-cfg	Release 7.2.1
Cisco-IOS-XR-um-multicast-routing-cfg	Release 7.2.1
Cisco-IOS-XR-um-object-group-cfg	Release 7.2.1
Cisco-IOS-XR-um-policymap-classmap-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-igmp-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-pim-cfg	Release 7.2.1
Cisco-IOS-XR-um-statistics-cfg	Release 7.2.1
Cisco-IOS-XR-um-ethernet-services-access-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-l2transport-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv4-prefix-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv6-prefix-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-amt-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-mld-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-msdp-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-bgp-cfg	Release 7.1.1
Cisco-IOS-XR-um-mpls-te-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-isis-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-ospf-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-ospfv3-cfg	Release 7.1.1
Cisco-IOS-XR-um-grpc-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-bundle-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-ethernet-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-ip-address-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-vrf-cfg	Release 7.0.1
Cisco-IOS-XR-um-interface-cfg	Release 7.0.1
Cisco-IOS-XR-um-mpls-l3vpn-cfg	Release 7.0.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-netconf-yang-cfg	Release 7.0.1
Cisco-IOS-XR-um-router-rib-cfg	Release 7.0.1
Cisco-IOS-XR-um-router-static-cfg	Release 7.0.1
Cisco-IOS-XR-um-snmp-server-cfg	Release 7.0.1
Cisco-IOS-XR-um-telemetry-model-driven-cfg	Release 7.0.1
Cisco-IOS-XR-um-vrf-cfg	Release 7.0.1
Cisco-IOS-XR-um-arp-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-arp-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-mpls-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-tunnel-cfg	Release 7.0.1
Cisco-IOS-XR-um-mpls-ldp-cfg	Release 7.0.1
Cisco-IOS-XR-um-mpls-lsd-cfg	Release 7.0.1
Cisco-IOS-XR-um-rsvp-cfg	Release 7.0.1
Cisco-IOS-XR-um-traps-mpls-ldp-cfg	Release 7.0.1