Programmability Configuration Guide for Cisco NCS 540 Series Routers, Cisco IOS XR Release 24.1.x, 24.2.x, 24.3.x

First Published: 2024-03-14

Last Modified: 2024-09-04

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

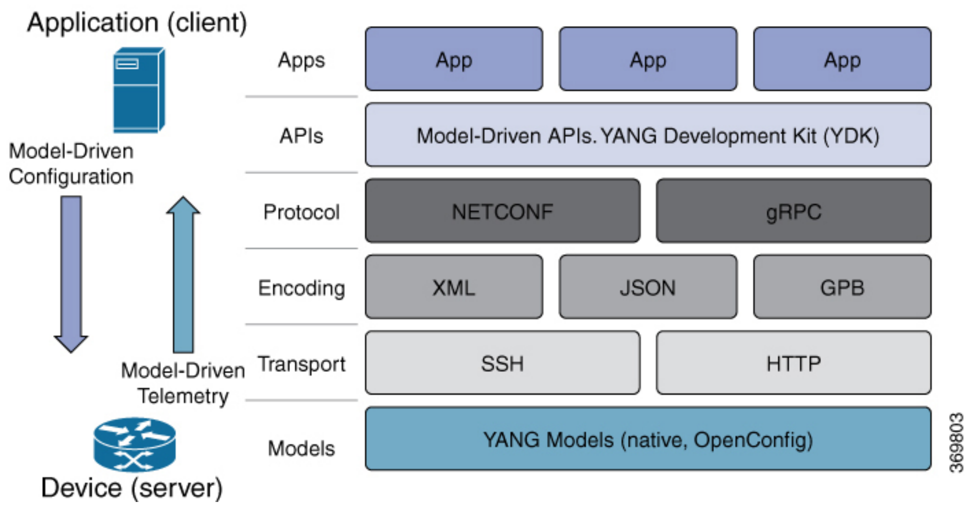
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the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

- **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.

YANG models can be: 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.

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 LLDP Data Model

The Link Layer Discovery Protocol (LLDP) data model is represented in the following structure:

```
$ cat Cisco-IOS-XR-ethernet-lldp-cfg.yang
module Cisco-IOS-XR-ethernet-lldp-cfg {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns"+
    "/yang/Cisco-IOS-XR-ethernet-lldp-cfg";


  prefix "ethernet-lldp-cfg";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import cisco-semver { prefix "semver"; }

  import Cisco-IOS-XR-ifmgr-cfg { prefix "a1"; }

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";

  contact
    "Cisco Systems, Inc.
     Customer Service

     Postal: 170 West Tasman Drive
     San Jose, CA 95134

     Tel: +1 800 553-NETS

     E-mail: cs-yang@cisco.com";

  description
    "This module contains a collection of YANG definitions
     for Cisco IOS-XR ethernet-lldp package configuration.

     This module contains definitions
     for the following management objects:
       lldp: Enable LLDP, or configure global LLDP subcommands

     This YANG module augments the
       Cisco-IOS-XR-ifmgr-cfg
     module with configuration data.

     Copyright (c) 2013-2019 by Cisco Systems, Inc.
     All rights reserved.";

  revision "2019-04-05" {
    description
      "Establish semantic version baseline.";
    semver:module-version "1.0.0";
  }
```

```
  revision "2017-05-01" {
    description
      "Fixing backward compatibility error in module.";
  }

  revision "2015-11-09" {
    description
      "IOS XR 6.0 revision.";
  }


  container lldp {
    description "Enable LLDP, or configure global LLDP subcommands";

    container tlv-select {
      presence "Indicates a tlv-select node is configured.";
      description "Selection of LLDP TLVs to disable";

      container system-name {
        description "System Name TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable System Name TLV";
        }
      }

      container port-description {
        description "Port Description TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable Port Description TLV";
        }
      }
.......................... (snipped) ..........................
      container management-address {
        description "Management Address TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable Management Address TLV";
        }
      }
      leaf tlv-select-enter {
        type boolean;
        mandatory true;
        description "enter lldp tlv-select submode";
      }
    }
    leaf holdtime {
      type uint32 {
        range "0..65535";
      }
      description
        "Length  of time  (in sec) that receiver must
        keep this packet";

  .......................... (snipped) ..........................
  }

  augment "/a1:interface-configurations/a1:interface-configuration" {
```

```
    container lldp {
      presence "Indicates a lldp node is configured.";
      description "Disable LLDP TX or RX";
.......................... (snipped) ...........................
    description
      "This augment extends the configuration data of
      'Cisco-IOS-XR-ifmgr-cfg'";
  }
}
```

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.

LLDP Configuration Data Model

The following example shows the LLDP interface manager configuration model in tree format.

```
module: Cisco-IOS-XR-ethernet-lldp-cfg
    +--rw lldp
       +--rw tlv-select!
       |  +--rw system-name
       |  |  +--rw disable?   boolean
       |  +--rw port-description
       |  |  +--rw disable?   boolean
       |  +--rw system-description
       |  |  +--rw disable?   boolean
       |  +--rw system-capabilities
       |  |  +--rw disable?   boolean
       |  +--rw management-address
       |  |  +--rw disable?   boolean
       |  +--rw tlv-select-enter       boolean
       +--rw holdtime?               uint32
       +--rw enable-priority-addr?   boolean
       +--rw extended-show-width?    boolean
       +--rw enable-subintf?         boolean
       +--rw enable-mgmtintf?        boolean
       +--rw timer?                  uint32
       +--rw reinit?                 uint32
       +--rw enable?                 boolean
module: Cisco-IOS-XR-ifmgr-cfg
    +--rw global-interface-configuration
    |  +--rw link-status?   Link-status-enum
    +--rw interface-configurations
       +--rw interface-configuration* [active interface-name]
          +--rw dampening
          |  +--rw args?                 enumeration
          |  +--rw half-life?            uint32
          |  +--rw reuse-threshold?      uint32
          |  +--rw suppress-threshold?   uint32
          |  +--rw suppress-time?        uint32
          |  +--rw restart-penalty?      uint32
          +--rw mtus
          |  +--rw mtu* [owner]
          |     +--rw owner    xr:Cisco-ios-xr-string
          |     +--rw mtu      uint32
          +--rw encapsulation
          |  +--rw encapsulation?         string
          |  +--rw capsulation-options?   uint32
          +--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 ethernet-lldp-cfg:lldp!
            +--rw ethernet-lldp-cfg:transmit
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:receive
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:lldp-intf-enter    boolean
            +--rw ethernet-lldp-cfg:enable?            Boolean

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

LLDP Operational Data Model

The following example shows the Link Layer Discovery Protocol (LLDP) interface manager operational model in tree format.

```
$ pyang -f tree Cisco-IOS-XR-ethernet-lldp-oper.yang
module: Cisco-IOS-XR-ethernet-lldp-oper
    +--ro lldp
       +--ro global-lldp
       |  +--ro lldp-info
       |     +--ro chassis-id?            string
       |     +--ro chassis-id-sub-type?   uint8
       |     +--ro system-name?           string
       |     +--ro timer?                 uint32
       |     +--ro hold-time?             uint32
       |     +--ro re-init?               uint32
       +--ro nodes
          +--ro node* [node-name]
             +--ro neighbors
             |  +--ro devices
             |  |  +--ro device*

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

  notifications:
    +---n lldp-event
       +--ro global-lldp
       |  +--ro lldp-info
       |     +--ro chassis-id?            string
       |     +--ro chassis-id-sub-type?   uint8
       |     +--ro system-name?           string
       |     +--ro timer?                 uint32
       |     +--ro hold-time?             uint32
       |     +--ro re-init?               uint32
       +--ro nodes
          +--ro node* [node-name]
             +--ro neighbors
             |  +--ro devices
             |  |  +--ro device*
             |  |     +--ro device-id?        string
             |  |     +--ro interface-name?   xr:Interface-name
             |  |     +--ro lldp-neighbor*
             |  |        +--ro detail
             |  |        |  +--ro network-addresses
             |  |        |  |  +--ro lldp-addr-entry*
             |  |        |  |     +--ro address
 ......................... (snipped) ..........................
           +--ro interfaces
             |  +--ro interface* [interface-name]
             |     +--ro interface-name             xr:Interface-name
             |     +--ro local-network-addresses
             |     |  +--ro lldp-addr-entry*
             |     |     +--ro address
```

```
|      |      |  +--ro address-type?   Lldp-l3-addr-protocol
|      |      |  +--ro ipv4-address?   inet:ipv4-address
|      |      |  +--ro ipv6-address?   In6-addr
|      |      +--ro ma-subtype?   uint8
|      |      +--ro if-num?       uint32
|      +--ro interface-name-xr?         xr:Interface-name
|      +--ro tx-enabled?                uint8
|      +--ro rx-enabled?                uint8
|      +--ro tx-state?                  string
|      +--ro rx-state?                  string
|      +--ro if-index?                  uint32
|      +--ro port-id?                   string
|      +--ro port-id-sub-type?          uint8
|      +--ro port-description?          string
......................... (snipped) ..........................
```

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:

The YANG models configure a feature, retrieve the operational state of the router, and perform actions.

- **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

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

YANG Module Set

You can provide structured, protocol-driven access to a network management configuration and its state information using YANG models. By default, all YANG models (native and OpenConfig) are accessible. You can activate a desired module-set using the **yang-server module-set** command to access a specific set of YANG modules.

Configure YANG Module Set

To activate a specific set of YANG module, use the **yang-server module-set** command.

```
Router# config
Router(config)# yang-server module-set XR-only
Router# end
```

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**

Note Starting from Release 7.11.1, the command **yang-describe** in the Command Line Interface (CLI) is deprecated.

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

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

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.

The following table shows a list of actions. For the full list of supported actions, query the device or see the YANG Data Models Navigator.

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
```

Example: Install Action

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

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

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

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

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

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<install-add xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-spirit-install-act">
   <packagepath>http://10.105.227.154/install_repo/fretta/651/651_02</packagepath>
        <packagename>ncs540-k9sec-1.0.0.0-r632.x86_64.rpm</packagename>
        <packagename>ncs540-li-1.0.0.0-r632.x86_64.rpm</packagename>
        <packagename>ncs540-mcast-1.0.0.0-r632.x86_64.rpm</packagename>
        <packagename>ncs5500-mini-x.iso-6.5.1.02Incs540-mini-x.iso-6.3.2</packagename>
        <packagename>ncs540-mpls-1.0.0.0-r632.x86_64.rpm</packagename>
</install-add>
</rpc>
```

Restrictions for Install Action

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

CHAPTER 2

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.

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.

Cisco IOS XR supports NETCONF 1.0 and 1.1 programmable management interfaces that are handled using two separate interfaces. From IOS XR, Release 7.3.1, a client can choose to establish a NETCONF 1.0 or 1.1 session using an interface for both these formats. A NETCONF proxy process waits for the `hello` message from its peer. If the proxy does not receive a `hello` message within the timeout period, it sends a NETCONF 1.1 `hello` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability
--snip--
</capabilities>
<session-id>5</session-id>
</hello>]]>]]>
```

The following examples show the `hello` messages for the NETCONF versions:

```
netconf-xml agent listens on port 22
```

```
netconf-yang agent listens on port 830
```

Version 1.0 The NETCONF XML agent accepts the message.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
</capabilities>
</hello>
```

Version 1.1 The NETCONF YANG agent accepts the message.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.1</capability>
</capabilities>
</hello>
```

Using NETCONF 1.1, the RPC requests begin with `#<number>` and end with `##`. The number indicates how many bytes that follow the request.

Example:

```
#371
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <filter>
    <isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-oper">
      <instances>
        <instance>
          <neighbors/>
          <instance-name/>
        </instance>
      </instances>
    </isis>
  </filter>
</get>
```

```
</rpc>

##
```

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
```

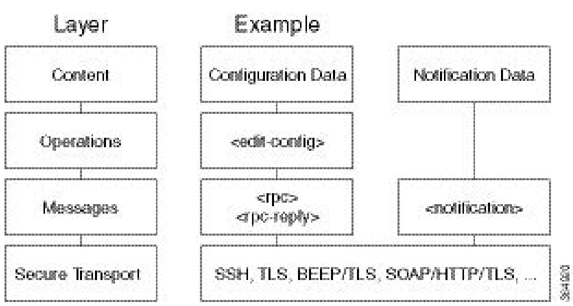
To enable the NETCONF SSH agent, use the following command:

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

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 compared to CLI.

- NETCONF Operations, on page 20
- Retrieve Default Parameters Using with-defaults Capability, on page 24
- Retrieve Transaction ID for NSO Operations, on page 30
- Set Router Clock Using Data Model in a NETCONF Session, on page 32

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</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>`

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

NETCONF Operation to Get Configuration

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

The client initiates a message to get the current configuration of LLDP running on the router. The router responds with the current LLDP 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>`
`  <lldp`
`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-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>`
	`  <lldp`
	`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-cfg">`
	
	`   <timer>60</timer>`
	`   <enable>true</enable>`
	`   <reinit>3</reinit>`
	`   <holdtime>150</holdtime>`
	`  </lldp>`
	
	` </data>`
	`</rpc-reply>`
	`319 bytes received`
	`6.409561ms elapsed`

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

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 router supports `<with-defaults>` capability to retrieve the default parameters of configuration and state data node using a NETCONF protocol operation. The `<with-defaults>` capability indicates which default-handling basic mode is supported by the server. It also indicates support for additional 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 the following command in Config mode:

```
netconf-yang agent
 ssh
 with-defaults-support enable
!
```

Once enabled, the capability is applied to all netconf-yang requests.

After enabling, the router must return the new capability as:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults:1.0?basic-mode=explicit
```

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

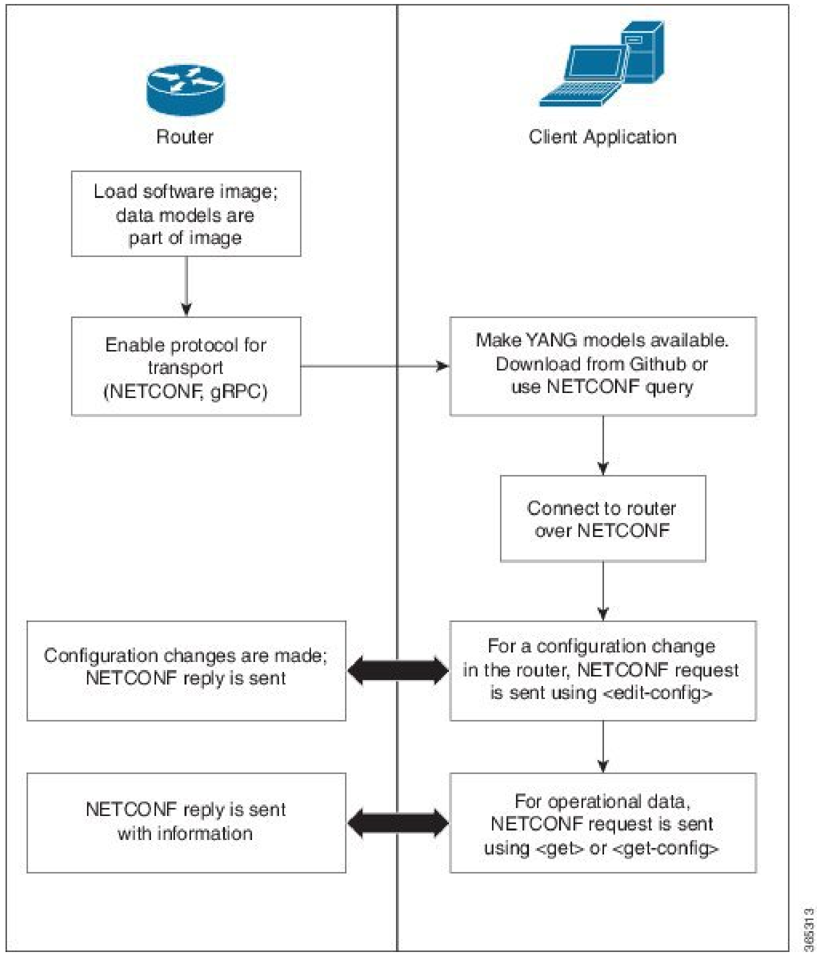
The process for using data models involves:

- 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*.

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

Figure 3: Process for Using Data Models

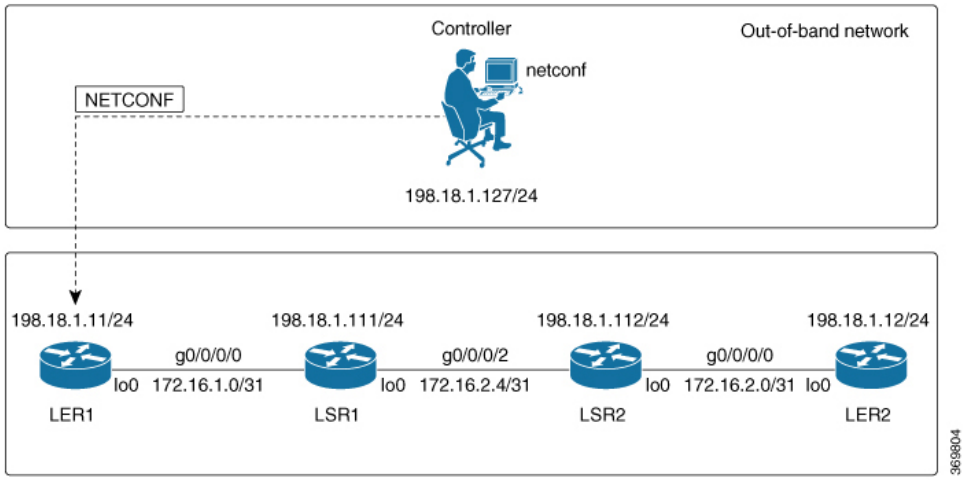


In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

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

---

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>
```

Step 5 Verify that the router clock is set to PST time zone.

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
<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 3

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.

• Configure Interfaces Using Data Models in a gRPC Session, on page 51

gRPC Operations

The following are the defined manageability service gRPC operations for Cisco IOS XR:

gRPC Operation	Description
GetConfig	Retrieves the configuration from the router.
GetModels	Gets the supported Yang models on the router
MergeConfig	Merges the input config with the existing device configuration.
DeleteConfig	Deletes one or more subtrees or leaves of configuration.
ReplaceConfig	Replaces part of the existing configuration with the input configuration.
CommitReplace	Replaces all existing configuration with the new configuration provided.
GetOper	Retrieves operational data.
CliConfig	Invokes the input CLI configuration.
ShowCmdTextOutput	Returns the output of a show command in the text form
ShowCmdJSONOutput	Returns the output of a show command in JSON form.

gRPC Operation to Get Configuration

This example shows how a gRPC GetConfig request works for LLDP feature.

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

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

<pre>rpc GetConfig
  {
   "Cisco-IOS-XR-cdp-cfg:cdp": [
    "cdp": "running-configuration"
    ]
  }
 	
rpc GetConfig
  {
   "Cisco-IOS-XR-ethernet-lldp-cfg:lldp": [
    "lldp": "running-configuration"
    ]
  }</pre>
	<pre>{
	 "Cisco-IOS-XR-cdp-cfg:cdp": {
	  "timer": 50,
	  "enable": true,
	  "log-adjacency": [
	   null
	  ],
	  "hold-time": 180,
	  "advertise-v1-only": [
	   null
	  ]
	 }
	}
	
	{
	"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": {
	  "timer": 60,
	  "enable": true,
	  "reinit": 3,
	  "holdtime": 150
	}
	}</pre>

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.

gRPC Network Operations Interface

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices. These services are to be used in conjunction with gRPC network management interface (gNMI) for all target state and operational state of a network. gNOI uses gRPC as the transport protocol and the configuration is same as that of gRPC. For more information about gNOI, see the Github repository.

Note This feature is not supported for the following PIDs:

- N540-ACC-SYS
- N540X-ACC-SYS (Premium)
- N540-24Z8Q2C-SYS

gRPC Network Security Interface

Table 4: Feature History Table

Feature Name	Release Information	Feature Description
gRPC Network Security Interface	Release 7.11.1	This release implements authorization mechanisms to restrict access to gRPC applications and services based on client permissions. This is made possible by introducing an authorization protocol buffer service for gRPC Network Security Interface (gNSI). Prior to this release, the gRPC services in the gNSI systems could be accessed by unauthorized users. This feature introduces the following change: **CLI**: • **gnsi load service authorization policy** • **show gnsi service authorization policy** To view the specification of gNSI, see Github repository.

gRPC Network Security Interface (gNSI) is a repository which contains security infrastructure services necessary for safe operations of an OpenConfig platform. The services such as authorization protocol buffer manage a network device's certificates and authorization policies.

This feature introduces a new authorization protocol buffer under gRPC gNSI. It contains gNSI.authz policies which prevent unauthorized users to access sensitive information. It defines an API that allows the configuration of the RPC service on a router. It also controls the user access and restricts authorization to update specific RPCs.

By default, gRPC-level authorization policy is provisioned using Secure ZTP. If the router is in zero-policy mode that is, in the absence of any policy, you can use gRPC authorization policy configuration to restrict access to specific users. The default authorization policy at the gRPC level can permit access to all RPCs except for the gNSI.authz RPCs.

If there is no policy specified or the policy is invalid, the router will fall back to zero-policy mode, in which the default behavior allows access to all gRPC services to all the users if their profiles are configured. If an invalid policy is configured, you can revert it by loading a valid policy using exec command **gnsi load service authorization policy.** For more information on how to create user profiles and update authorization policy for these user profiles, see How to Update gRPC-Level Authorization Policy, on page 43. Using **show gnsi service authorization policy** command, you can see the active policy in a router.

We have introduced the following commands in this release :

- **gnsi load service authorization policy**: To load and update the gRPC-level authorization policy in a router.
- **show gnsi service authorization policy**: To see the active policy applied in a router.

Note When both gNSI and gNOI are configured, gNSI takes precedence over gNOI. If niether gNSI nor gNOI is configured, then tls trsutpoint's data is considered for certificate management.

The following RPCs are used to perform key operations at the system level such as updating and displaying the current status of the authorization policy in a router.

Table 5: Operations

RPC	Description
gNSI.authz.Rotate()	Updates the gRPC-level authorization policy.
gNSI.authz.Probe()	Verifies the authenticity of a user based on the defined policy of the gRPC-level authorization policy engine.
gNSI.authz.Get()	Shows the current instance of the gRPC-level authorization policy, including the version and date of creation of the policy.

How to Update gRPC-Level Authorization Policy

gRPC-level authorization policy is configured by default at the time of router deployment using secure ZTP. You can update the same gRPC-level authorization policy using any of two the following methods:

- Using gNSI Client.
- Using exec command.

Updating the gRPC-Level Authorization Policy in the Router Using gNSI Client

Before you start

When a router boots for the first time, it should have the following prerequisites:

- The gNSI.authz service is up and running.
- The default gRPC-level authorization policy is added for all gRPC services.
- The default gRPC-level authorization policy allows access to all RPCs.

The following steps are used to update the gRPC-level authorization policy:

1. Initiate the **gNSI.authz.Rotate()** streaming RPC. This step creates a streaming connection between the router and management application (client).

Note Only one `gNSI.authz.Rotate()` must be in progress at a time. Any other RPC request is rejected by the server.

2. The client uploads new gRPC-level authorization policy using the **UploadRequest** message.

Note
- There must be only one gRPC-level authorization policy in the router. All the policies must be defined in the same gRPC-level authorization policy which is being updated. As `gNSI.authz.Rotate()` method replaces all previously defined or used policies once the **finalize** message is sent.
- The upgrade information is passed to the version and the created_on fields. These information are not used by the gNSI.authz service. It is designed to help you to track the active gRPC-level authorization policy on a particular router.

3. The router activates the gRPC-level authorization policy.
4. The router sends the `UploadResponse` message back to the client after activating the new policy.
5. The client verifies the new gRPC-level authorization policy using separate **gNSI.authz.Probe()** RPCs.
6. The client sends the **FinalizeRequest** message, indicating the previous gRPC-level authorization policy is replaced.

Note It is not recommended to close the stream without sending the **finalize** message. It results in the abandoning of the uploaded policy and rollback to the one that was active before the `gNSI.authz.Rotate()` RPC started.

Below is an example of a gRPC-level authorization policy that allows admins, V1,V2,V3 and V4, access to all RPCs that are defined by the gNSI.ssh interface. All the other users won't have access to call any of the gNSI.ssh RPCs:

```
{
  "version": "version-1",
  "created_on": "1632779276520673693",
  "policy": {
    "name": "gNSI.ssh policy",
    "allow_rules": [{
      "name": "admin-access",
      "source": {
        "principals": [
          "spiffe://company.com/sa/V1",
          "spiffe://company.com/sa/V2"
        ]
      },
      "request": {
        "paths": [
          "/gnsi.ssh.Ssh/*"
        ]
      }
    }],
    "deny_rules": [{
      "name": "sales-access",
```

```
      "source": {
        "principals": [
          "spiffe://company.com/sa/V3",
          "spiffe://company.com/sa/V4"
        ]
      },
      "request": {
        "paths": [
          "/gnsi.ssh.Ssh/MutateAccountCredentials",
          "/gnsi.ssh.Ssh/MutateHostCredentials"
        ]
      }
    }]
  }
}
```

Updating the gRPC-Level Authorization Policy file Using Exec Command

Use the following steps to update the authorization policy in the router.

1. Create the users profiles for the users who need to be added in the authorization policy. You can skip this step if you have already defined the user profiles.

   The following example creates three users who are added in the authorization policy.

   ```
   Router(config)#username V1
   Router(config-un)#group root-lr
   Router(config-un)#group cisco-support
   Router(config-un)#secret x
   Router(config-un)#exit
   Router(config)#username V2
   Router(config-un)#group root-lr
   Router(config-un)#password x
   Router(config-un)#exit
   Router(config)#username V3
   Router(config-un)#group root-lr
   Router(config-un)#password x
   Router(config-un)#commit
   ```

2. Enable **tls-mutual** to establish the secure mutual between the client and the router.

   ```
   Router(config)#grpc
   Router(config-grpc)#port 0
   Router(config-grpc)#tls-mutual
   Router(config-grpc)#certificate-authentication
   Router(config-grpc)#commit
   ```

3. Define the gRPC-level authorization policy.

   The following sample gRPC-level authorization policy defines authorization policy for the users V1, V2 and V3.

   ```
   {
       "name": "authz",
       "allow_rules": [
           {
               "name": "allow all gNMI for all users",
               "source": {
                   "principals": [
                       "*"
                   ]
   ```

```
            },
            "request": {
                "paths": [
                    "*"
                ]
            }
        }
    ],
    "deny_rules": [
         {
            "name": "deny gNMI set for oper users",
            "source": {
                "principals": [
                    "V1"
                ]
            },
            "request": {
                "paths": [
                    "/gnmi.gNMI/Get".
                ]
            }
        },

        {
            "name": "deny gNMI set for oper users",
            "source": {
                "principals": [
                    "V2"
                ]
            },
            "request": {
                "paths": [
                    "/gnmi.gNMI/Get"
                ]
            }
        },
        {
            "name": "deny gNMI set for oper users",
            "source": {
                "principals": [
                    "V3"
                ]
            },
            "request": {
                "paths": [
                    "/gnmi.gNMI/Set"
                ]
            }
        }
    ]
}
```


4. Copy the gRPC-level authorization policy to the router.

   The following example copies the gNSI Authz policy to the router:

```
-bash-4.2$ scp test.json V1@192.0.2.255:/disk0:/
Password:
test.json
                                   100%  993   161.4KB/s   00:00
-bash-4.2$
```

5. Activate the gRPC-level authorization policy to the router.

The following example loads the policy to the router.

```
Router(config)#gnsi load service authorization policy /disk0:/test.json
Successfully loaded policy
```

Verification

Use the **show gnsi service authorization policy** to verify if the policy is active in the router.

```
Router#show gnsi service authorization policy
Wed Jul 19 10:56:14.509 UTC{
    "version": "1.0",
    "created_on": 1700816204,
    "policy": {
        "name": "authz",
        "allow_rules": [
            {
                "name": "allow all gNMI for all users",
                "request": {
                    "paths": [
                        "*"
                    ]
                },
                "source": {
                    "principals": [
                        "*"
                    ]
                }
            }
        ],
        "deny_rules": [
            {
                "name": "deny gNMI set for oper users",
                "request": {
                    "paths": [
                        "/gnmi.gNMI/*"
                    ]
                },
                "source": {
                    "principals": [
                        "User1"
                    ]
                }
            }
        ]
    }
}
```

In the following example, User1 user tries to access the **get** RPC request for which the permission is denied in the above authorization policy.

```
bash-4.2$ ./gnmi_cli -address 198.51.100.255 -ca_crt
certs/certs/ca.cert -client_crt certs/certs/User1.pem -client_key
certs/certs/User1.key -server_name ems.cisco.com -get -proto get-oper.proto
```

Output

```
E0720 14:49:42.277504   26473 gnmi_cli.go:195]
target returned RPC error for Get("path:{origin:"openconfig-interfaces"
elem:{name:"interfaces"}
 elem:{name:"interface" key:{key:"name" value:"HundredGigE0/0/0/0"}}}
 type:OPERATIONAL encoding:JSON_IETF"):
rpc error: code = PermissionDenied desc = unauthorized RPC request rejected
```

gRPC Authentication Modes

gRPC supports the following authentication modes to secure communication between clients and servers. These authentication modes help ensure that only authorized entities can access the gRPC services, like gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device will authenticate the user and perform various authorization checks to validate the user.

The following table lists the authentication type and configuration requirements:

Table 6: gRPC Authentication Modes and Configuration Requirements

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Metadata with TLS	username, password	username	**grpc**	username, password, and CA
Metadata without TLS	username, password	username	**grpc no-tls**	username, password
Metadata with Mutual TLS	username, password	username	**grpc tls-mutual**	username, password, client certificate, client key, and CA
Certificate based Authentication	client certificate's common name field	username from client certificate's common name field	**grpc tls-mutual** and **grpc certificate authentication**	client certificate, client key, and CA

Certificate based Authentication

In Extensible Manageability Services (EMS) gRPC, the certificates play a vital role in ensuring secure and authenticated communication. The EMS gRPC utilizes the following certificates for authentication:

```
/misc/config/grpc/ems.pem
/misc/config/grpc/ems.key
/misc/config/grpc/ca.cert
```

Note For clients to use the certificates, ensure to copy the certificates from **/misc/config/grpc/**

Generation of Certificates

These certificates are typically generated using a Certificate Authority (CA) by the device. The EMS certificates, including the server certificate (**ems.pem**), public key (**ems.key**), and CA certificate (**ca.cert**), are generated with specific parameters like the common name **ems.cisco.com** to uniquely identify the EMS server and placed in the **/misc/config/grpc/** location.

The default certificates that are generated by the server are Server-only TLS certificates and by using these certificates you can authenticate the identity of the server.

Usage of Certificates

These certificates are used for enabling secure communication through Transport Layer Security (TLS) between gRPC clients and the EMS server. The client should use **ems.pem** and **ca.cert** to initiate the TLS authentication.

To update the certificates, ensure to copy the new certificates that has been generated earlier to the location and restart the server.

Custom Certificates

If you want to use your own certificates for EMS gRPC communication, then you can follow a workflow to generate a custom certificates with the required parameters and then configure the EMS server to use these custom certificates. This process involves replacing the default EMS certificates with the custom ones and ensuring that the gRPC clients also trust the custom CA certificate. For more information on how to customize the **common-name**, see *Certificate Common-Name For Dial-in Using gRPC Protocol*.

Certificate Common-Name For Dial-in Using gRPC Protocol

Table 7: Feature History Table

Feature Name	Release Information	Description
Certificate Common-Name For Dial-in Using gRPC Protocol	Release 24.1.1	You can now specify a **common-name** for the certificate generated by the router while using gRPC dial-in. Earlier, the **common-name** in the certificate was fixed as *ems.cisco.com* and was not configurable. Using a specified **common-name** avoids potential certification failures where you may specify a hostname different from the fixed common name to connect to the router. The feature introduces these changes: **CLI:** • **grpc certificate common-name** **YANG Data Model:** • New XPath for `Cisco-IOS-XR-um-grpc-cfg.yang` • New XPath for `Cisco-IOS-XR-man-ems-cfg` (see GitHub, YANG Data Models Navigator)

When using gRPC dial-in on Cisco IOS-XR router, the **common-name** associated with the certificate generated by the router is fixed as *ems.cisco.com* and this caused failure during certificate verification.

From Cisco IOS XR Release 24.1.1, you can now have the flexibility of specifying the common-name in the certificate using the **grpc certifcate common-name** command. This allows gRPC clients to verify if the domain name in the certificate matches the domain name of the gRPC server being accessed.

Configure Certificate Common Name For Dial-in

Configure a common name to be used in EMSD certificates for gRPC dial-in.

Step 1 Configure a common name.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#certificate common-name cisco.com
Router(config-grpc)#commit
```

Use the show command to verify the common name:

```
Router#show grpc
Certificate common name                : cisco.com
```

Note For the above configuration to be successful, ensure to regenerate the certificate. so that the new EMSD certificates include the configured common name.

To **regenerate** the self-signed certificate, perform the following steps.

Step 2 Remove the certificates: `/misc/config/grpc/ems.pem`, `/misc/config/grpc/ems.key`, and `/misc/config/grpc/ca.cert` from `/misc/config/grpc` file.

Example:

```
Router#run ls -ltr /misc/config/grpc/

total 16
drwx------. 2 root root 4096 Feb 14 09:17 dialout
-rw-rw-rw-. 1 root root 1505 Feb 14 10:58 ems.pem
-rw-------. 1 root root 1675 Feb 14 10:58 ems.key
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert

Router#run rm -rf /misc/config/grpc/ems.pem /misc/config/grpc/ems.key

Router#run ls -ltr /misc/config/grpc/

total 8
drwx------. 2 root root 4096 Feb 14 09:17 dialout
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
```

Step 3 Restart gRPC server by toggling the TLS configuration.

Configure gRPC with non TLS and then re-configure with TLS.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#no-tls
Router(config-grpc)#commit

Router#run ls -ltr /misc/config/grpc/

total 8
```

```
drwx------. 2 root root 4096 Feb 14 09:17 dialout
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert

Router#config
Router(config)#grpc
Router(config-grpc)#no no-tls
Router(config-grpc)#commit

Router#run ls -ltr /misc/config/grpc/

total 16
drwx------. 2 root root 4096 Feb 14 09:17 dialout
-rw-rw-rw-. 1 root root 1505 Feb 14 14:23 ems.pem
-rw-------. 1 root root 1675 Feb 14 14:23 ems.key
-rw-r--r--. 1 root root 1505 Feb 14 14:23 ca.cert
```

Copy the newly generated `/misc/config/grpc/ems.pem` certificate in this path (from the device) to the gRPC client.

Configure Interfaces Using Data Models in a gRPC Session

Google-defined remote procedure call () 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.

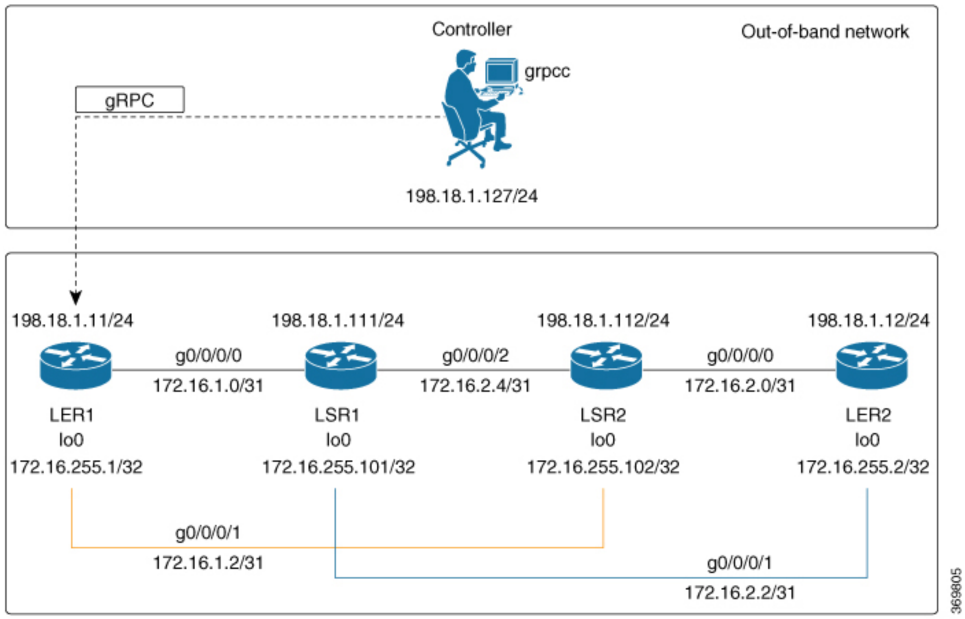
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network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

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
- 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.

Step 1 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.

a) 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.

b) Set the session parameters.

Example:

```
Router(config)#grpc {address-family | certificate-authentication | dscp | max-concurrent-streams
 | max-request-per-user | max-request-total | max-streams |
max-streams-per-user | no-tls | tlsv1-disable | tls-cipher | tls-mutual | tls-trustpoint |
service-layer | vrf}
```

where:

- `address-family`: set the address family identifier type.
- `certificate-authentication`: enables certificate based authentication
- `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
- `tlsv1-disable`: disable TLS version 1.0
- `service-layer`: enable the grpc service layer configuration.

  This parameter is not supported in Cisco ASR 9000 Series Routers, Cisco NCS560 Series Routers, , and Cisco NCS540 Series Routers.
- `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.

Step 2 Configure the 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 40. 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.

a) 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 ...
```

b) Configure a loopback0 interface on LER1.

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"
       }
      }
     }
    }
   ]
 }
}
```

c) 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 '
'
```

d) Configure the ethernet interface 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"
      }
     }
    }
   }
  ]
 }
}
```

e) 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 '
'
```

f) 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 ''
```

Step 3 Verify that the loopback interface and the ethernet interface on router LER1 are operational.

Example:

```
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 4

Enhancements to Data Models

This section provides an overview of the enhancements made to data models.

Improved YANG Input Validator and Get Requests

Table 8: Feature History Table

Feature Name	Release Information	Description
Improved YANG Input Validator and Get Requests	Release 7.10.1	The OpenConfig data models provide a structure for managing networks via YANG protocols. With this release, enhancements to the configuration architecture improve input validations and ensure that the Get requests made through gNMI or NETCONF protocols return only explicitly configured OpenConfig leaves. Previously, Get requests returned all the items in the Cisco native data models that the system could convert into OpenConfig items, regardless of whether they were initially configured via OpenConfig. We have added a new legacy mode option for a limited number of releases which helps you preserve this behaviour.

In IOS XR Software Release 7.10.1, the following are the enhancements to improve YANG Input Validator and Get Requests:

- Get requests made via NETCONF or gNMI now return only OpenConfig leaves that were configured using OpenConfig models.

  Use the legacy mode as follows:

  NETCONF: Add a legacy mode attribute to the **get-config** request tag,

  Example: **get-config xmlns:xr-md="http://cisco.com/ns/yang/cisco-xr-metadata" xr-md:mode="legacy"**

  gNMI: Set the origin to **openconfig-legacy**.

- Improved input validation for OpenConfig configurations to provide a more consistent experience across the schema.

  The new validation includes enhanced error reporting, though some errors may include references to XR configuration schema paths and item values in the message string.

- OpenConfig leaves now return default values consistently.

  Get requests use the **Explicit Basic Mode** (refer RFC6243) to return only the OpenConfig leaves that were explicitly configured.

Usage Guidelines and Limitations

In this release, the following usage guidelines and limitations apply based on the following functionalities:

- Upgrades to Cisco IOS XR Software Release 7.10.1 and later will not show OpenConfig leaves in Get requests until OpenConfig has been successfully committed.
- Similarly, downgrading from Release 7.10.1 to an earlier version and then upgrading back to Release 7.10.1 will not show OpenConfig leaves in Get requests until OpenConfig has been successfully committed.
- Each feature must be fully configured using OpenConfig or Cisco native data model or CLI.

  If configuration items applied to a feature via OpenConfig are overridden by configuring those items directly via Cisco native data model, this will not be reflected in the system view of currently configured OpenConfig items.

  Use the Cisco native data model to configure features not supported by OpenConfig data model.

- Use either gNMI or NETCONF to manage configuration via OpenConfig. We recommend not to use both the management agents on the same device simultaneously.

  Once a successful commit has been made using gNMI or NETCONF, that management agent is considered the **active agent**.

  OpenConfig items cannot be configured by the non-active agent. However, the non-active agent can configure Cisco native data model items and perform Get requests on any configuration items.

  All OpenConfig leaves must first be removed by the active agent before a different agent can be used.

- During the commit process (which can take many minutes for large changesets), Get requests can be made on the running datastore.

  Other request types like, Edit request, Commit request from other clients, and Get request on the candidate datastore of another client are rejected.

- When ACLs are configured via OpenConfig, CLI actions such as resequencing ACLs and copying ACLs will not be reflected in the system view of the current OpenConfig configuration.

- Configuration modifications made by Config Scripts to features configured through OpenConfig will not be reflected in the system view of the current OpenConfig configuration which is returned from Get-config operations.
- Configuration removal from the system may occur as a result of some events, such as install operations and startup configuration failures during line card insertion.

  OpenConfig items currently configured do not reflect this change. In such cases, a syslog will be generated to remind the user to manually apply OpenConfig configurations to the system.
- All OpenConfig will be removed from the system when a **Commit Replace** operation is performed using the CLI.
- By using the **show running-config | (xml | json) openconfig** command, you can still view the running OpenConfig. However, you cannot filter the view using XR CLI configuration keywords.
- The **load rollback changes** and **load commit changes** commands are not supported for rollback or commit that include OpenConfig leaves.

OpenConfig Data Model Enhancements

Table 9: Feature History Table

Feature Name	Release Information	Description
Revised OpenConfig MPLS Model to Version 3.0.1 for Streaming Telemetry	Release 7.3.3	The OpenConfig MPLS data model provides data definitions for Multiprotocol Label Switching (MPLS) configuration and associated signaling and traffic engineering protocols. In this release, the following data models are revised for streaming telemetry from OpenConfig version 2.3.0 to version 3.0.1: • openconfig-mpls • openconfig-mpls-te • openconfig-mpls-rsvp • openconfig-mpls-igp • openconfig-mpls-types • openconfig-mpls-sr You can access this data model from the Github repository.

OAM for MPLS and SR-MPLS in mpls-ping and mpls-traceroute Data Models

Table 10: Feature History Table

Feature Name	Release Information	Description
YANG Data Models for MPLS OAM RPCs	Release 7.3.2	This feature introduces the `Cisco-IOS-XR-mpls-ping-act` and `Cisco-IOS-XR-mpls-traceroute-act` YANG data models to accommodate operations, administration and maintenance (OAM) RPCs for MPLS and SR-MPLS. You can access these Cisco IOS XR native data models from the Github repository.

The Cisco-IOS-XR-mpls-ping-act and Cisco-IOS-XR-mpls-traceroute-act YANG data models are introduced to provide the following options:

- Ping for MPLS:
    - MPLS IPv4 address
    - MPLS TE
    - FEC-129 Pseudowire
    - FEC-128 Pseudowire
    - Multisegment Pseudowire
- Ping for SR-MPLS:
    - SR policy name or BSID with LSP end-point
    - SR MPLS IPv4 address
    - SR Nil-FEC labels
    - SR Flexible Algorithm
- Traceroute for MPLS:
    - MPLS IPv4 address
    - MPLS TE
- Traceroute for SR-MPLS:
    - SR policy name or BSID with LSP end-point

- SR MPLS IPv4 address
- SR Nil-FEC labels
- SR Flexible Algorithm

The following example shows the ping operation for an SR policy and LSP end-point:

```
<mpls-ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-ping-act">
  <sr-mpls>
  <policy>
    <name>srte_c_10_ep_10.10.10.1</name>
      <lsp-endpoint>10.10.10.4</lsp-endpoint>
  </policy>
  </sr-mpls>
  <request-options-parameters>
    <brief>true</brief>
  </request-options-parameters>
</mpls-ping>
```

Response:

```
<?xml version="1.0"?>
 <mpls-ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-ping-act">
  <request-options-parameters>
   <exp>0</exp>
   <fec>false</fec>
   <interval>0</interval>
   <ddmap>false</ddmap>
   <force-explicit-null>false</force-explicit-null>
   <packet-output>
    <interface-name>None</interface-name>
    <next-hop>0.0.0.0</next-hop>
   </packet-output>
   <pad>abcd</pad>
   <repeat>5</repeat>
   <reply>
    <dscp>255</dscp>
    <reply-mode>default</reply-mode>
    <pad-tlv>false</pad-tlv>
   </reply>
   <size>100</size>
   <source>0.0.0.0</source>
   <destination>127.0.0.1</destination>
   <sweep>
    <minimum>100</minimum>
    <maximum>100</maximum>
    <increment>1</increment>
   </sweep>
   <brief>true</brief>
   <timeout>2</timeout>
   <ttl>255</ttl>
  </request-options-parameters>
  <replies>
   <reply>
    <reply-index>1</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>2</reply-index>
```

```
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>3</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>4</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>5</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
  </replies>
 </mpls-ping-response>
```

The following example shows the ping operation for an SR policy BSID and LSP end-point:

```
<mpls-ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-ping-act">
<sr-mpls>
<policy>
    <bsid>1000</bsid>
    <lsp-endpoint>10.10.10.4</lsp-endpoint>
</policy>
</sr-mpls>
<request-options-parameters>
    <brief>true</brief>
</request-options-parameters>
</mpls-ping>
```

Response:

```
<?xml version="1.0"?>
 <mpls-ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-ping-act">
  <request-options-parameters>
   <exp>0</exp>
   <fec>false</fec>
   <interval>0</interval>
   <ddmap>false</ddmap>
   <force-explicit-null>false</force-explicit-null>
   <packet-output>
    <interface-name>None</interface-name>
    <next-hop>0.0.0.0</next-hop>
   </packet-output>
   <pad>abcd</pad>
   <repeat>5</repeat>
   <reply>
    <dscp>255</dscp>
    <reply-mode>default</reply-mode>
    <pad-tlv>false</pad-tlv>
```

```
   </reply>
   <size>100</size>
   <source>0.0.0.0</source>
   <destination>127.0.0.1</destination>
   <sweep>
    <minimum>100</minimum>
    <maximum>100</maximum>
    <increment>1</increment>
   </sweep>
   <brief>true</brief>
   <timeout>2</timeout>
   <ttl>255</ttl>
  </request-options-parameters>
  <replies>
   <reply>
    <reply-index>1</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>2</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>3</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>4</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
   <reply>
    <reply-index>5</reply-index>
    <return-code>3</return-code>
    <return-char>!</return-char>
    <reply-addr>14.14.14.3</reply-addr>
    <size>100</size>
   </reply>
  </replies>
 </mpls-ping-response>
```

The following example shows the traceroute operation for an SR policy and LSP end-point:

```
<mpls-traceroute xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-traceroute-act">
<sr-mpls>
<policy>
    <name>srte_c_10_ep_10.10.10.1</name>
    <lsp-endpoint>10.10.10.4</lsp-endpoint>
</policy>
</sr-mpls>
<request-options-parameters>
    <brief>true</brief>
</request-options-parameters>
```

```
</mpls-traceroute>
```

Response:


```
<?xml version="1.0"?>
 <mpls-traceroute-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-mpls-traceroute-act">

  <request-options-parameters>
   <exp>0</exp>
   <fec>false</fec>
   <ddmap>false</ddmap>
   <force-explicit-null>false</force-explicit-null>
   <packet-output>
    <interface-name>None</interface-name>
    <next-hop>0.0.0.0</next-hop>
   </packet-output>
   <reply>
    <dscp>255</dscp>
    <reply-mode>default</reply-mode>
   </reply>
   <source>0.0.0.0</source>
   <destination>127.0.0.1</destination>
   <brief>true</brief>
   <timeout>2</timeout>
   <ttl>30</ttl>
  </request-options-parameters>
  <paths>
   <path>
    <path-index>0</path-index>
    <hops>
    <hop>
        <hop-index>0</hop-index>
        <hop-origin-ip>11.11.11.1</hop-origin-ip>
        <hop-destination-ip>11.11.11.2</hop-destination-ip>
        <mtu>1500</mtu>
        <dsmap-label-stack>
            <dsmap-label>
                <label>16003</label>
            </dsmap-label>
        </dsmap-label-stack>
        <return-code>0</return-code>
        <return-char> </return-char>
    </hop>
    <hop>
        <hop-index>1</hop-index>
        <hop-origin-ip>11.11.11.2</hop-origin-ip>
        <hop-destination-ip>14.14.14.3</hop-destination-ip>
        <mtu>1500</mtu>
        <dsmap-label-stack>
            <dsmap-label>
                <label>3</label>
            </dsmap-label>
        </dsmap-label-stack>
        <return-code>8</return-code>
        <return-char>L</return-char>
    </hop>
    <hop>
        <hop-index>2</hop-index>
        <hop-origin-ip>14.14.14.3</hop-origin-ip>
        <hop-destination-ip></hop-destination-ip>
        <mtu>0</mtu>
        <dsmap-label-stack/>
        <return-code>3</return-code>
        <return-char>!</return-char>
```

```
    </hop>
    </hops>
   </path>
  </paths>
  </mpls-traceroute-response>
```

Automatic Resynchronization of OpenConfig Configuration

Table 11: Feature History Table

Feature Name	Release Information	Feature Description
View Inconsistent OpenConfig Configuration	Release 24.1.1	OpenConfig infrastructure now provides an operational data YANG model, `Cisco-IOS-XR-yiny-oper`, which can be queried to view the inconsistent OpenConfig configuration caused due to activities such as interface breakout operations, installation activities or insertion of a new line card. See GitHub, YANG Data Models Navigator
Automatic Resynchronization of OpenConfig Configuration	Release 7.11.1	OpenConfig infrastructure can now reapply all the OpenConfig configurations automatically if there are any discrepancies in the running configuration. With this feature, there is no need for manual replacement of the OpenConfig configuration using Netconf or gNMI. The re-sync operation is triggered if the running configurations and the OpenConfig configuration go out of sync after any system event that removes some running configurations from the system. A corresponding system log gets generated to indicate the re-sync status.

In the earlier releases, when activities such as interface breakout operations, installation activities or insertion of a new line card took place, there was a risk of OpenConfig configuration and the running configuration going out of sync. A full replacement of the OpenConfig configuration was required in order to get the OpenConfig configurations back in sync using Netconf or gNMI.

From the Cisco IOS XR Software Release 7.11.1, if the OpenConfig configurations and running configurations go out of sync, or any activities takes place which may result in the two configurations to go out of sync, the system automatically reapplies all the OpenConfig configurations and resolve the sync issue. If there is a synchronization issue between the running configuration and the OpenConfig configuration, a corresponding system log is generated to indicate it. Similarly, a corresponding system log is generated indicating the status of the re-synchronization attempt.

This feature is enabled by default. This process is completely automated.

From the Cisco IOS XR Software Release 24.1.1, the new `Cisco-IOS-XR-yiny-oper` YANG model displays the OpenConfig configuration which is out of sync with the running configuration, including the error associated with each out of sync configuration.

The `Cisco-IOS-XR-yiny-oper` operational data is a snapshot of the current system status, rather than a record of all past failures. That is, if an item of configuration is out of sync and is later resolved, such as through a resynchronization or another configuration operation, then this configuration is no longer considered out of sync and is removed from the snapshot.

Operations that Remove Running Configuration

Here are three types of operation that can have the effect of removing running configuration from the system. Running configurations are either affected because they directly remove configuration in the system or because they result in configuration failing to be accepted by the system during start-up.

- **Install operations**: Running configuration can be removed during non-reload and reload install operations. During non-reload install, running configuration is removed when it is incompatible with the new software. In this case, it is directly removed by the Install infra. The configuration is removed during reload install operations if the attempt to restore the startup configuration is partially successful.
- **Breakout interfaces configuration**: When breakout interfaces are configured or de-configured, all the existing configuration on interfaces is affected. The affect may be creation or deletion of the parent and child interfaces. This results in an inconsistency between the running configuration and the OpenConfig datastore for any of the removed configurations that was mapped from OpenConfig configuration.

  The automatic restoration of OpenConfig configuration resolves this inconsistency by re-adding that removed configuration.
- **New line card insertion**: On insertion of a new line card into the system, any pre-configuration for that card is verified for the first time and may be rejected, causing it to be removed. This results in an inconsistency between the running configuration and the OpenConfig datastore.

In any of the above scenarios, if there is a sync issue, system logs are generated and the system tries to reapply all the OpenConfig configurations. If the re-sync attempt is successful, the configurations which were removed earlier, are re-applied. If the re-sync attempt fails, this means that some of the OpenConfig configuration is no longer valid.

Note The above scenarios are invalid if there are no OpenConfig configuration present in the system.

System Logs Indicating Out-of-Sync Configuration

System log messages are generated due to the above operations that can lead to discrepancies in configurations on the router. Listed are examples of system log messages raised if any such discrepancies occur.

Table 12: Examples of system log messages generated due to Out-of-Sync Configurations :

Event Name Displayed in the System Log	Description
unexpected commit errors	When an unexpected commit errors in case of a SysDB server crash.
config rollback (to a commit ID created using a different software version)	When a configuration rollbacks back to a commit ID created using a different software version.
inconsistent configuration	This system log is generated when an inconsistency alarm is raised due to failure in restoring the start-up configurations after activities like system reload or insertion of a new line card. Re-synchronization of the configuration is triggered only after the alarm is cleared.
configuration removal (triggered on 0/2/CPU0 by the last config operation for interface GigabitEthernet0/2/0/0 and 6 other interfaces)	When interface configuration is removed in response to a change in interface breakout configuration.
configuration removal (to prepare for an install operation)	Configuration is removed from the system during a non-reload install operation due to incompatibility with the new software.

Alarms Related to Out-of-Sync OpenConfig Configuration

- **Inconsistency alarm**: When a there is a failure in restoring the start-up configurations after a system reload or insertion of a new line card, inconsistency alarm is raised. If the inconsistency alarm is raised, you can see an informational system log is generated which indicates that the OpenConfig configuration and running configuration may be out of sync. A re-sync attempt will be made when the configuration inconsistency alarm is cleared. This system log is an early warning that the system is potentially out of sync.

  Inconsistency alarm message:

  ```
  NMI OpenConfig configuration is potentially out of sync with the running configuration
  (details: system configuration become inconsistent during OIR restore on 0/0/CPU0). An
  automatic reapply of the OpenConfig configuration will be performed when the inconsistency
  alarm is cleared.
  ```

- **Missing item in the OpenConfig datastore alarm**: If there are missing items in the configurations which could not be added to the OpenConfig datastore while loading in a snapshot from disk, you can see an error system log is raised which indicates that there are some items which are absent in the running OpenConfig configuration. This scenario occurs when the yang schema is changed from the time the snapshot was created.

  Item missing alarm message:

  ```
  gNMI OpenConfig configuration is potentially out of sync with the running configuration:
  3 failed to be applied to the system (details: snapshot 2 was created with a different
  schema version). The system may contain config items mapped from OC that no longer exist
  in the OC datastore. Automatic attempts to reapply OC will not remove these items, even
  if they otherwise succeed. Config should be replaced manually using a GNMI Replace
  operation.
  ```

System Logs Generated During Configuration Resynchronization:

When an attempt to re-apply OpenConfig (resynchronization) is complete, the following informational system logs are generated to indicate the user that the OpenConfig and running configuration were out of sync, and whether the attempt to resolve this was successful.

- Successful re-sync:

  ```
  As a result of configuration removal (to prepare for an install operation), the gNMI
  OpenConfig configuration has been successfully reapplied.
  ```

- Unsuccessful re-sync:

  ```
  As a result of configuration removal (to prepare for an install operation), an attempt
  to reapply the gNMI OpenConfig configuration was made, but some items remain out of
  sync with the running configuration. Out of sync configuration can be viewed using the
  Cisco-IOS-XR-yiny-oper model.
  ```

- Re-sync failure during mapping of OpenConfig configurations to XR configurations:

  ```
  As a result of configuration removal (to prepare for an install operation), the attempt
  to reapply the gNMI OpenConfig configuration failed, and the out of sync configuration
  could not be updated. gNMI OpenConfig configuration is potentially out of sync with the
  running configuration. Configuration should be reapplied manually using a GNMI Replace
  operation
  ```

Re-sync failure during mapping of OpenConfig configurations to XR configurations is a rare scenario. When there is a failure in the re-sync process while mapping the OpenConfig configuration to XR items, it causes the re-sync request to aborted. This scenario is only possible after an install which changes the OpenConfig mappings such that some configuration is no longer supported.

Resolve Out of Sync Configuration

An automatic resynchronization fails if the out-of-sync scenario is unresolved or the OpenConfig configuration and running XR configuration are out of sync.

Here are the two scenarios with steps to resolve the out-of-sync configuration if an attempt for automatic resynchronization fails.

Resync Fails Partially:

1. Query the items of configuration which are out of sync using the `Cisco-IOS-XR-yiny-oper` YANG model
2. For each out-of-sync configuration item:
    - Delete the OpenConfig items that are out of sync.
    - Re-add the deleted OpenConfig items in a separate request.

Resync Fails Completely:

Perform a full replace of the OpenConfig configuration using Netconf or gNMI.

By successfully completing these steps, you can now ensure that all configurations are in sync.

YANG Model Data for Inconsistent Configuration

Each configuration of the `Cisco-IOS-XR-yiny-oper` YANG model has a list entry with the following fields:

- **Path**: The path of the XR configuration, in YPath format.
- **Input paths**: The OpenConfig paths of the items from which the XR configuration is mapped.

  Activity: If last occurrence of this failure was:

  - in a user-initiated commit operation.
  - in a system-initiated resynchronization attempt, after an install operation, breakout interfaces being configured, or line card insertion.

- **Operation**: If a configuration being `set` or `delete`:

  For a configuration that is out of sync because it failed during a resynchronization attempt, the operation is always `set`, but for a user-initiated commit operation, the operation is whichever the user was attempting during the commit.

- **Latest failure type**: If the latest failure is a `verify` failure or an `apply` failure.

  Only `verify` errors are currently tracked as out of sync and reported in the operational data, but this field is present in the model for potential future usage if `apply` errors are also tracked.

  - For configuration that fails during startup, both `verify` and `apply` failures can make the configurations out of sync.
  - For configuration that fails during a commit operation, only `apply` failures can make the configuration out of sync. This is because configuration is not allowed in the datastore if `verify` failures occur during a commit operation.

- **Latest error**: The latest error message describing the error.

CHAPTER 5

Unified Configuration Models

Table 13: Feature History Table

Feature Name	Release Information	Description
Unified Data Model to map script file to the custom OID	Release 7.5.3	Use the `Cisco-IOS-XR-um-script-server-cfg.yang` unified data model to map script file to the custom OID.
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.

Table 14: Unified Models

Unified Models	Introduced in Release
Cisco-IOS-XR-um-script-server-cfg	Release 7.5.3
Cisco-IOS-XR-um-script-cfg	Release 7.5.3
Cisco-IOS-XR-um-if-ipsubscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-session-redundancy-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-accounting-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-redundancy-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-opendns-cfg	Release 7.5.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-dynamic-template-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-cfg	Release 7.5.1
Cisco-IOS-XR-um-lpts-profiling-cfg	Release 7.5.1
Cisco-IOS-XR-um-ppp-cfg	Release 7.5.1
Cisco-IOS-XR-um-pppoe-cfg	Release 7.5.1
Cisco-IOS-XR-um-vpdn-cfg	Release 7.5.1
Cisco-IOS-XR-um-aaa-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-ipv4-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-ipv6-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-vrf-cfg	Release 7.5.1
Cisco-IOS-XR-um-mibs-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-monitor-session-cfg	Release 7.5.1
Cisco-IOS-XR-um-l2tp-class-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-dhcpv6d-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-service-policy-cfg	Release 7.5.1
Cisco-IOS-XR-um-snmp-server mroutemib send-all-cfg	Release 7.5.1
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

Unified Models	Introduced in Release
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
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

Unified Models	Introduced in Release
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
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

Unified Models	Introduced in Release
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
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

Unified Models	Introduced in Release
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
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

Unified Models	Introduced in Release
Cisco-IOS-XR-um-traps-mpls-ldp-cfg	Release 7.0.1

PART I

Automation Scripts

CHAPTER 6

Achieve Network Operational Simplicity Using Automation Scripts

Network automation is imperative to deploy and manage the networks with large-scale cloud-computing architectures. The automation can be achieved through standard model-driven data models. To cater to the automation requirements, you leverage the Cisco IOS XR infrastructure to make API calls and run scripts from an external controller. These off-box scripts take advantage of the exposed interfaces such as NETCONF, SNMP, SSH to work on the network element. However, there is need to maintain an external controller to interact with the router.

To simplify the operational infrastructure, the automation scripts can be run on the router, eliminating the need for an external controller. The execution of the different types of scripts are faster and reliable as it is not dependent on the speed or network reachability of the external controller. Most script types interact with IOS XR Software using standard protocols such as NETCONF. You can download script to the router, configure scripts, view operational data, and set responses to events in the router.

In summary, on-box scripting is similar to off-box scripting, with the exception that the management software that runs in an external controller is now part of the router software. The scripts programmatically automate configuration and operational tasks on the network devices. You can create customized scripts that are based on your network requirement and execute scripts on routers running Cisco IOS XR operating system. The packages that support scripting are provided in the software image.

Note You can create scripts using Python 3.5.

Explore the Types of Automation Scripts

There are four types of on-box automation scripts that you can leverage to automate your network operations:

- Configuration (Config) scripts
- Execution (Exec) scripts
- Process scripts
- EEM scripts

The following table provides the scope and benefit of on-box scripts:

Table 15: On-Box Automation Scripts

	Config Scripts	**Exec Scripts**	**Process Scripts**	**EEM Scripts**
What is the scope of the script?	Enforce contextual and conditional changes to configurations, validate configurations before committing the changes to detect and notify potential errors. If configuration does not comply with the rules that are defined in the script, an action can be invoked. For example, generate a warning, syslog message, or halt a commit operation.	Run operational commands or RPCs, process the output, generate syslogs, configure system, perform system action commands such as system reload, process restarts, and collect logs for further evaluation.	Daemonize to continuously run as an agent on the router to execute additional checks outside traditional ZTP. Daemonized scripts are similar to exec scripts but run continuously. The script executes operational commands on the router and analyzes the output.	Run operational commands or RPCs, generate, and determine the next steps like logging the root cause or changing device configuration. Event policies can upload the output of event scripts to an on-box or off-box location for further analysis.
How to invoke the script?	All config scripts are processed automatically when **commit** command is executed on the router.	Exec script is invoked manually via CLI command or RPC.	Process script is activated via configuration CLI command.	Event scripts are invoked by defined event policies in response to a system event and allow for immediate action to take effect.
What are the main benefits of using the script?	Simplifies complex configurations and averts potential errors before a configuration is committed. Ensures that the network configuration complies with rules and policies that are defined in the script.	Collects operational information, and decreases the time that is involved in troubleshooting issues. Provides flexibility in changing the input parameters for every script run. This fosters dynamic automation of operational information.	Runs scripts as a daemon to continuously perform tasks that are not transient.	Automates log collection upon detecting error conditions that are defined by event policies. Uploads the output of event scripts to an on-box or off-box location for further analysis.

CHAPTER 7

EEM Scripts

Table 16: Feature History Table

Feature Name	Release Information	Description
EEM Scripts	Release 24.3.1	Embedded Event Manager (EEM), a Cisco IOS XR software component, tracks and monitors events on your Cisco device and then executes specific predefined actions. You can create an action using Python scripts and trigger it when a specified event occurs. Using EEM and Python, you can automate tasks, build small functionalities, and create workarounds. The EEM Scripts have the advantage of executing the scripts on the local device, eliminating the need to use an external scripting engine or monitoring device.

Cisco IOS XR Embedded Event Manager (EEM) scripts are also known as event scripts that are triggered automatically in response to events on the router. An event can be any significant occurrence, not limited to errors, that has happened within the system. You can use these scripts to detect issues in the network in real time, program certain conditions in response to the event, detect and generate an action when those conditions are met, and execute policy (script) when an event is generated. The script acts in response to the events and reduces the troubleshooting time involved in resolving the issues. For example, you can enforce LACP dampening if a bundle interface has flapped 5 times in less than 30 secs, and define the script to disable the interface for 2 minutes.

You can programmatically define the event and actions separately and map them using a policy map via CLI or NETCONF RPCs. Whenever the configured event occurs, the action that is mapped to it is executed. The same event and action can be mapped to multiple policy maps. You can map the same event and action in 64 policy maps, and add a maximum of 5 different actions in a policy map.

You can create event scripts using Python 3.5 programming language. For the list of supported Python packages. You can also configure the EEM policies using Tool Command Language (TCL) scripts. To know more about

TCL scripts, see *Configuring and Managing Embedded Event Manager Policies* Chapter in System Monitoring Configuration Guide.

This chapter gets you started with provisioning your Python automation scripts on the router.

Note This section does not delve into creating Python scripts, but assumes that you have basic understanding of Python programming language. This section will walk you through the process involved in deploying and using the scripts on the router.

Workflow to Run Event Scripts

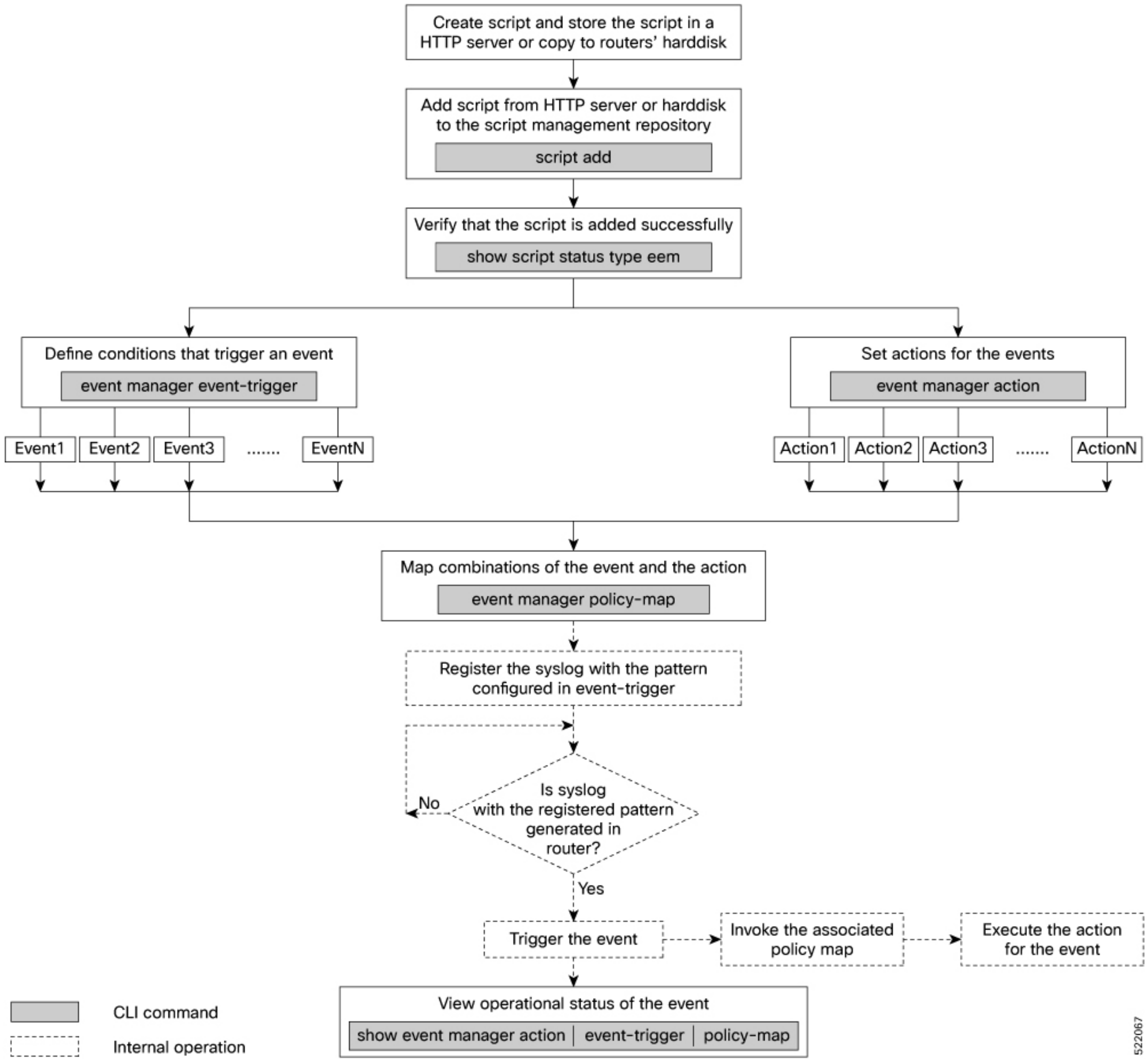
Complete the following tasks to provision eem scripts:

- Download the script—Store the eem script on an HTTP server or copy to the harddisk of the router. Add the eem script from the HTTP server or harddisk to the script management repository on the router using the **script add eem** command.
- Define events—Configure the events with the trigger conditions using the **event manager event-trigger** command.
- Define actions to the events—Setup the actions that must be performed in response to an event using **event manager action** command.
- Create policy map—Put together the events and the actions in a policy map using **event manager policy-map** command.

  Note An eem script is invoked automatically when the event occurs. With the event, the event-trigger invokes the corresponding policy-map to implement the actions in response to the event.

- View operational status of the event—Retrieve the operational data using the **show event-manager action | event-trigger | policy-map** command.

The following image shows a workflow diagram representing the steps involved in using an event script:



The following sections cover the steps to run event scripts:

1. Download the Script to the Router
2. Define Trigger Conditions for an Event
3. Create Actions for Events
4. Create a Policy Map of Events and Actions
5. View Operational Status of Event Scripts

Download the Script to the Router

To manage the scripts, you must add the scripts to the script management repository on the router. A subdirectory is created for each script type. By default, this repository stores the downloaded scripts in the appropriate subdirectory based on script type.

Script Type	Download Location
config	`harddisk:/mirror/script-mgmt/config`
exec	`harddisk:/mirror/script-mgmt/exec`
process	`harddisk:/mirror/script-mgmt/process`
eem	`harddisk:/mirror/script-mgmt/eem`

The scripts are added to the script management repository using two methods:

- **Method 1:** Add script from a server
- **Method 2:** Copy script from external repository to harddisk using **scp** or **copy** command

In this section, you learn how to add `eem-script.py` script to the script management repository.

Step 1 Add the script to the script management repository on the router using one of the two options:

- **Add Script From a Server**

  Add the script from a configured HTTP server or the harddisk location in the router.

```
Router#script add eem <script-location> <script.py>
```

  The following example shows a process script `eem-script.py` downloaded from an external repository `http://192.0.2.0/scripts`:

```
Router#script add eem http://192.0.2.0/scripts eem-script.py
Fri Aug 20 05:03:40.791 UTC
eem-script.py has been added to the script repository
```

  You can add a maximum of 10 scripts simultaneously.

```
Router#script add eem <script-location> <script1.py> <script2.py> ... <script10.py>
```

  You can also specify the checksum value while downloading the script. This value ensures that the file being copied is genuine. You can fetch the checksum of the script from the server from where you are downloading the script. However, specifying checksum while downloading the script is optional.

```
Router#script add eem http://192.0.2.0/scripts eem-script.py checksum SHA256 <checksum-value>
```

  For multiple scripts, use the following syntax to specify the checksum:

```
Router#script add eem http://192.0.2.0/scripts <script1.py> <script1-checksum> <script2.py>
<script2-checksum>
... <script10.py> <script10-checksum>
```

  If you specify the checksum for one script, you must specify the checksum for all the scripts that you download.

  Note Only SHA256 checksum is supported.

- **Copy the Script from an External Repository**

  You can copy the script from the external repository to the routers' harddisk and then add the script to the script management repository.

  a. Copy the script from a remote location to harddisk using scp or copy command.

```
Router#scp userx@192.0.2.0:/scripts/eem-script.py /harddisk:/
```

b. Add the script from the harddisk to the script management repository.

```
Router#script add eem /harddisk:/ eem-script.py
Fri Aug 20 05:03:40.791 UTC
eem-script.py has been added to the script repository
```

Step 2 Verify that the scripts are downloaded to the script management repository on the router.

Example:

```
Router#show script status
Wed Aug 25 23:10:50.453 UTC
==========================================================================================
 Name               | Type      | Status            | Last Action | Action Time
------------------------------------------------------------------------------------------
 eem-script.py      | eem       | Config Checksum   | NEW         | Tue Aug 24 10:44:53 2021
==========================================================================================
```

Script `eem-script.py` is copied to `harddisk:/mirror/script-mgmt/eem` directory on the router.

Define Trigger Conditions for an Event

You define the event, and create a set of instructions that trigger a match to this event. You can create multiple events.

Before you begin

Ensure that the script is added to the script management repository..

Step 1 Register the event.

Example:

```
Router(config)#event manager event-trigger eventT10
```

You can configure more options to trigger an event:

Keyword	Description
occurrence	Number of occurrences before the event is raised. **Note** The **occurrence** keyword is supported only for syslog events.
period	Time interval during which configured occurrence should take place. **Note** The **period** keyword is supported only for syslog events.

<table>
<tr><th>Keyword</th><th>Description</th></tr>
<tr><td>type</td><td>Configure the type of event.
Rate limit—Configure rate limit in seconds or milliseconds. After the event is triggered, the event trigger does not happen even if the event occurs any number of times, till this time has elapsed.
Syslog—Configure syslog pattern, severity.
Timer—Configure watch dog timer in seconds; cron timer as a text string with five fields separated by a space.
Track—Configure event-trigger for track (object tracking), track state (<code>UP</code>, <code>DOWN</code>, or <code>ANY</code>). If event-trigger is configured for track state <code>UP</code>, then it gets triggered when the track state changes from <code>DOWN</code> to <code>UP</code>, and vice-versa.
Telemetry—Define events based on telemetry data. With this feature, you can perform the following operations:
a. Monitor any operational state such as interface status, and trigger an action when the state changes to a specific value.
b. Monitor any counter or statistics in an operational data, and trigger an action when it reaches a threshold.
c. Monitor rate of change of any operational attribute, and trigger an action based on threshold.
Note exact match supported on string and threshold or rate limit is supported only for integer type telemetry data
Configure sensor path for exact match, threshold or rate depending on the telemetry data type. The exact match is supported on string data type, and threshold and rate limit is supported only for interger data type. Use the following command to verify the sensor path or query before configuring the event trigger.
<code>Router#event manager telemetry sensor-path</code>
<code><sensor-path> json-query <query></code>
It is mandatory to enable model-driven telemetry using the command:
<code>Router#telemetry model-driven</code></td></tr>
</table>

Step 2 Configure the type for the event.

Example

Example: The following example shows the configuration for syslog event type. If severity is configured, the event gets triggered only if both the syslog severity and the syslog pattern match with the syslog generated on the router. If severity is not configured, it is set to `all`, where only pattern match is considered for the event to trigger.

```
Router(config)#event manager event-trigger eventT10
 type syslog pattern "<pattern-to-match>" severity <value>
```

```
Router(config)#event manager event-trigger eventT10
rate-limit seconds <time-in-seconds>
type syslog pattern "<pattern-to-match>" severity <value>
```

The severity values are:

```
alert      Syslog priority 1
critical   Syslog priority 2
debug      Syslog priority 7 (lowest)
emergency  Syslog priority 0 (highest)
error      Syslog priority 3
info       Syslog priority 6
notice     Syslog priority 5
warning    Syslog priority 4
```

The following example shows a syslog pattern `L2-BM-6-ACTIVE` with severity value critical:

```
Router(config)#event manager event-trigger eventT10
 type syslog pattern "L2-BM-6-ACTIVE" severity info
```

The event gets triggered, if both the syslog pattern `L2-BM-6-ACTIVE` and severity value `info` match.

Create Actions for Events

Define the actions that must be taken when an event occurs.

Before you begin

Ensure that the following prerequisites are met before you configure the action:

-
- Define Trigger Conditions for an Event, on page 89

Step 1 Set the event action.

Example:

```
Router(config)#event manager action action1
```

Step 2 Define the type of action. For example, the action is a Python script.

Example:

```
Router(config)#event manager action action1 type script action1.py
```

Step 3 Configure the maximum run time of the script for the event.

Example:

```
Router(config)#event manager action action1 type script action1.py maxrun seconds 30
```

The default value is 20 seconds.

Step 4 Configure the checksum for the script. This configuration is mandatory. Every script is associated with a checksum hash value. This value ensures the integrity of the script, and that the script is not tampered. The checksum is a string of numbers and letters that act as a fingerprint for script.

a) Retrieve the SHA256 checksum hash value for the script from the IOS XR Linux bash shell.

Example:

```
Router#run
[node0_RP0_CPU0:~]$sha256sum /harddisk:/mirror/script-mgmt/eem/action1.py
407ce32678a5fc4b0ad49e83acad6453ad1d47e8dad9501cf139daa75d53e3dd
/harddisk:/mirror/script-mgmt/eem/action1.py
```

b) Configure the checksum for the script.

Example:

```
Router(config)#event manager action action1 type script action1.py checksum
sha256 407ce32678a5fc4b0ad49e83acad6453ad1d47e8dad9501cf139daa75d53e3dd
```

Step 5 Enter the username for the script to execute.

Example:

```
Router(config)#event manager action action1 username eem_user
```

Create a Policy Map of Events and Actions

Create a policy to map events and actions. You can configure a policy that associates multiple actions with an event or use the same action with different events. The policy can be triggered if an event or multiple events occur at a specified number of times within a specified period of time. The occurrence and period are optional parameters. You can add multiple events to a policy-map with boolean (AND or OR) correlation. EEM triggers the script when the correlation defined in the policy-map for the events is true. For example, a multi-event policy-map for event1 and event2 with event1 AND event2 boolean operation is triggered only when both event1 and event2 are true.

Before you begin

Ensure that the following prerequisites are met before you create a policy map:

- Define Trigger Conditions for an Event, on page 89
- Create Actions for Events, on page 91

Step 1 Create a policy map.

Example:

Note Ensure that the operations when configuring multiple events are within double quotes `""`.

where,

- occurrence: Specifies the number of times the total correlation occurs before an EEM event is raised. If occurrence is not specified, the policy-map gets triggered on every occurrence of the event. The occurance vale ranges from 1 to 32. An occurrence that is configured with multiple events is considered as only one occurrence if the boolean logic operations becomes true.
- period: Time interval in seconds, during which the event occurs. The period must be an integer number between 1 to 429496729 seconds.

Step 2 Define the action that must be implemeted when the event occurs. Maximum of 5 actions can be mapped to a policy map.

Example:

```
Router(config-policy-map)#action action1
```

Step 3 Configure the name of the event to trigger the policy-map.

Example:

```
Router(config-policy-map)#trigger event event10
```

The following example shows the policy-map for multiple events:

```
event manager policy-map policy001
 trigger multi-event "event1 OR (event4 AND event2)"
 period 60
 action action2
 occurrence 2
!
```

View Operational Status of Event Scripts

Retrieve the operational status of events, actions and policy maps.

Before you begin

Ensure that the following prerequisites are met before you trigger the event:

- Define Trigger Conditions for an Event, on page 89
- Create Actions for Events, on page 91
- Create a Policy Map of Events and Actions, on page 92

Step 1 Run the **show event manager event-trigger all** command to view the summary of basic data of all events that are configured.

Example:

```
Router#show event manager event-trigger all
Tue Aug 24 14:47:35.803 IST
Thu May 20 20:41:03.690 UTC
No. Name    esid   Type    Occurs  Period  Trigger-Count Policy-Count Status
1   event1  1008   syslog  2       1800    4             1            active
2   event2  1009   syslog  2       1800    4             1            active
3   event3  1010   syslog  2       1800    4             1            active
4   event4  1011   syslog  2       1800    4             1            active
5   event5  1012   syslog  2       1800    4             1            active
6   event6  1013   syslog  2       1800    4             1            active
7   event7  1014   syslog  2       1800    4             1            active
8   event8  1015   syslog  2       1800    4             1            active
9   event9  1016   syslog  2       1800    4             1            active
```

Use the **show event manager event-trigger all detailed** command to view the details about the match criteria that you configured, severity level, policies mapped to the events and so on.

Use the **show event manager event-trigger <event-name> detailed** command to view the details about the individual events.

Step 2 Run the **show event manager policy-map all** command to view the summary of all the configured policy maps.

Example:

```
Router#show event manager policy-map all
Tue Aug 24 14:48:52.153 IST
No. Name      Occurs   period    Trigger-Count     Status
1   policy1   NA       NA        1                 active
2   policy2   NA       NA        1                 active
3   policy3   NA       NA        1                 active
4   policy4   NA       NA        1                 active
```

Use the **show event manager policy-map all detailed** command to view the details about mapping of associated events and actions in the policy maps.

Use the **show event manager policy-map <policy-map-name> detailed** command to view the details about the individual policy maps.

Step 3 Run the **show event manager action <action-name> detailed** commad to view the details of an action.

Example:

```
Router#show event manager action action1 detailed
Tue Aug 24 16:05:44.298 UTC

Action name: action1
Action type: script
EEM Script name: event_script_1.py
Action triggered count: 1
Action policy count: 1
Username: eem_user
Checksum: 407ce32678a5fc4b0ad49e83acad6453ad1d47e8dad9501cf139daa75d53e3dd
Last execution status: Success

Policy mapping info
1    action1                       policy1
```

Use the **show event manager action all** and **show event manager action all detailed** command to view the summary and details about all the configured actions.

Example: Shut Inactive Bundle Interfaces Using EEM Script

In this example, you use an EEM event to look for a syslog message and trigger a Python script. The script does two things:

- Triggers an event on the interface inactive log as part of Bundle-Ether1, and shuts down the interface.
- Runs the **show tech-support bundles** command to collect debug data.

Step 1 Create an eem script `event_script_action_bundle_shut.py`. Store the script on an HTTP server or copy the script to the harddisk of the router.

Example:

```
from iosxr.xrcli.xrcli_helper import *
from cisco.script_mgmt import xrlog
```

```
logger = xrlog.getScriptLogger('sample_script')
syslog = xrlog.getSysLogger('sample_script')
helper = XrcliHelper(debug = True)

syslog.info('Execution of event manager action script event_script_action_bundle_shut.py started')

config = """interface Bundle-Ether1
shutdown"""

cmd = "show tech-support bundles"

if __name__ == '__main__':
  res = helper.xr_apply_config_string(config)
  if res['status'] == 'success':
     syslog.info('OPS_EVENT_SCRIPT_ACTION : Configuration succeeded')
  else:
     syslog.error('OPS_EVENT_SCRIPT_ACTION : Configuration failed')

  res = helper.xrcli_exec(cmd)
  if res['status'] == 'success':
     syslog.info('OPS_EVENT_SCRIPT_ACTION : show tech started')
  else:
     syslog.error('OPS_EVENT_SCRIPT_ACTION : show tech failed')

  syslog.info('Execution of event manager action script event_script_action_bundle_shut.py ended')
```

Step 2 Add the script from HTTP server or harddisk to the script management repository..

Step 3 After the configured type matches the syslog pattern, the script is triggered in response to the detected event. You can view the running configuration for the event manager.

Example:

```
Router#show running-config event manager
Mon Aug 30 06:23:32.974 UTC
event manager action action1
 username eem_user
  type script script-name eem_script_bundle_shut.py maxrun seconds 600 checksum sha256
2386d8f71b2d6f6f6e77a7a39d3b4d38cca07f9eaf2a4de7cd40c1b027a4e248
!
event manager policy-map policy1
 trigger event event1
  action action1
!
 event manager event-trigger event1
  type syslog pattern "%L2-BM-6-ACTIVE : FortyGigE0/0/0/13 is no longer Active as part of Bundle-Ether1"
!
```