Programmability Configuration Guide for Cisco 8000 Series Routers, IOS XR Release 24.1.x, 24.2.x , 24.3.x

First Published: 2024-03-01

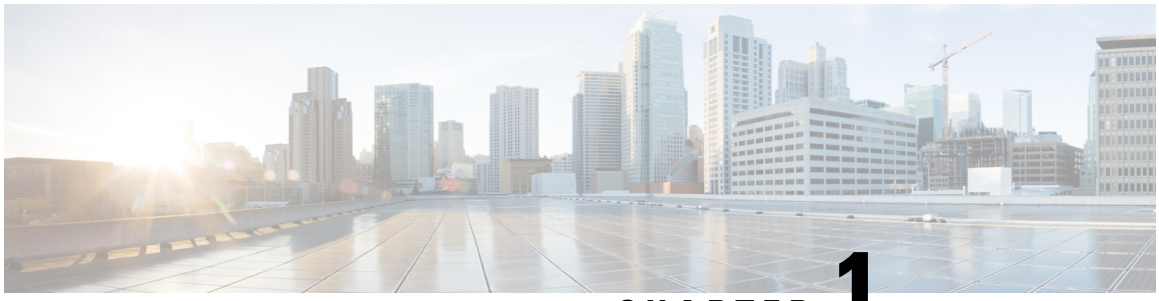
Last Modified: 2024-09-01

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

CONTENTS

PART I

YANG Data Models



CHAPTER 1

New and Changed Feature Information

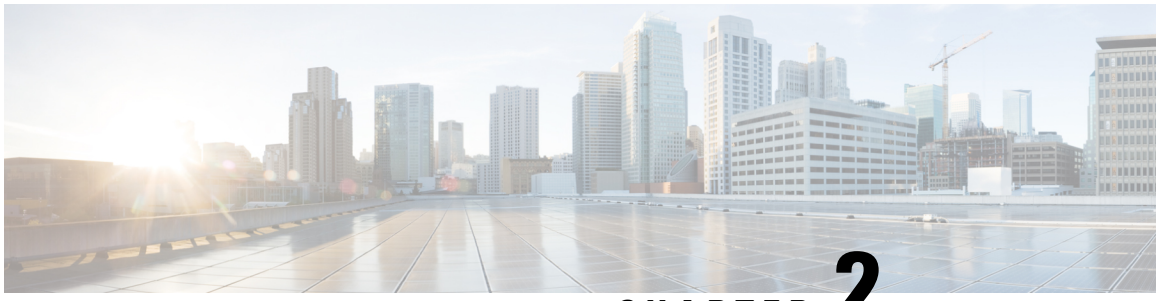
This section lists all the new and changed features for the Programmability Configuration Guide.

Programmability Features Added or Modified in IOS XR Release 24.x.x

Table 1: New and Changed Programmability Features

Feature	Description	Changed in Release	Where Documented
gNSI Accounting Logging	This feature was introduced	Release 24.3.1	gNSI Acctz Logging, on page 93
TPM Enrollment and Attestation	This feature was introduced	Release 24.3.1	TPM Enrollment and Attestation, on page 125
SPIFFE ID-Based Authentication and Authorization Services for gRPC Services	This feature was introduced	Release 24.2.11	SPIFFE ID-Based Authentication and Authorization Services for gRPC Services, on page 51
gNMI Union Replace Operation	This feature was introduced	Release 24.2.11	gNMI Union Replace Operation, on page 66
gNMI XPath-Based Authorization	This feature was introduced	Release 24.2.11	gNMI XPath-Based Authorization, on page 71
gNOI Packet Link Qualification	This feature was introduced	Release 24.2.11	gNOI Packet Link Qualification, on page 85

Feature	Description	Changed in Release	Where Documented
gNSI Rotate Credentials Update	This feature was introduced	Release 24.2.11	gNSI Credentialz Update, on page 97
NETCONF Version 1.0 with YANG Support	This feature was introduced	Release 24.2.11	NETCONF Version 1.0 with YANG Support, on page 42
Preprogram Backup LSPs Using Service Layer API	This feature was introduced	Release 24.2.11	Preprogram Backup LSPs Using Service Layer API, on page 124
Manage certificates using Certz.proto	This feature was introduced	Release 24.1.1	Manage certificates using Certz.proto, on page 102
Set Limit on Concurrent Streams for gRPC Server	This feature was introduced	Release 24.1.1	Set Limit on Concurrent Streams for gRPC Server
IANA Port Numbers For gRPC Services	This feature was introduced	Release 24.1.1	IANA Port Numbers For gRPC Services
View Inconsistent OpenConfig Configuration	This feature was introduced	Release 24.1.1	View Inconsistent OpenConfig Configuration



CHAPTER 2

YANG Data Models for Programmability Features

This chapter provides information about the YANG data models for Programmability features.

- Using YANG Data Models, on page 3

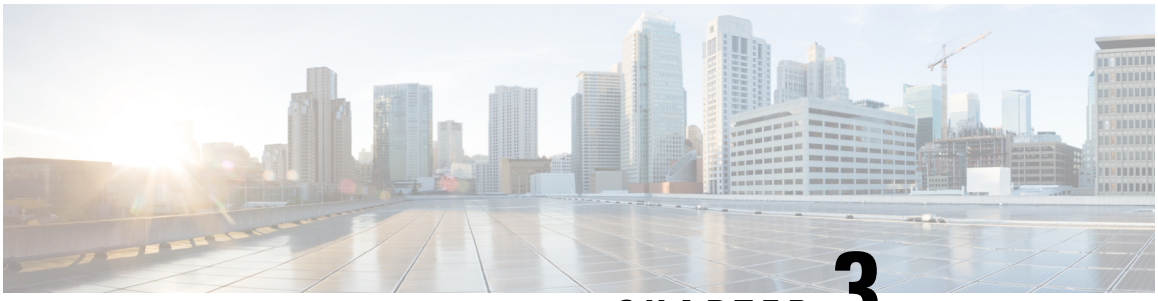
Using YANG Data Models

Cisco IOS XR supports a programmatic way of configuring and collecting operational data of a network device using YANG data models. Although configurations using CLIs are easier and human-readable, automating the configuration using model-driven programmability results in scalability.

The data models are available in the release image, and are also published in the Github repository. Navigate to the release folder of interest to view the list of supported data models and their definitions. Each data model defines a complete and cohesive model, or augments an existing data model with additional XPaths. To view a comprehensive list of the data models supported in a release, navigate to the **Available-Content.md** file in the repository.

You can also view the data model definitions using the YANG Data Models Navigator tool. This GUI-based and easy-to-use tool helps you explore the nuances of the data model and view the dependencies between various containers in the model. You can view the list of models supported across Cisco IOS XR releases and platforms, locate a specific model, view the containers and their respective lists, leaves, and leaf lists presented visually in a tree structure. This visual tree form helps you get insights into nodes that can help you automate your network.

To get started with using the data models, see the *Programmability Configuration Guide*.



CHAPTER 3

Drive Network Automation Using Programmable YANG Data Models

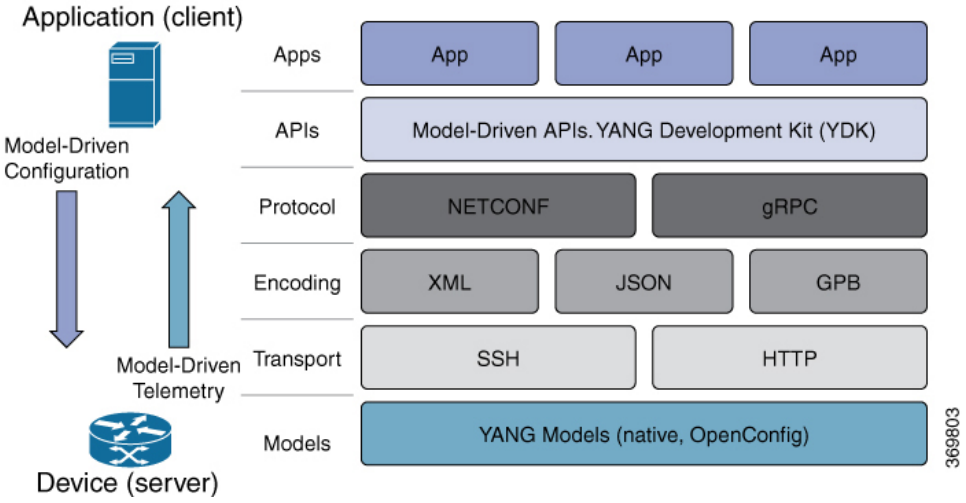
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the 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, on page 6
- Access the Data Models, on page 13
- CLI to Yang Mapping Tool, on page 14
- Prevent Partial Pseudo-Atomic Committed Configurations, on page 16
- Communication Protocols, on page 17
- YANG Actions, on page 18

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 2: 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
```

Prevent Partial Pseudo-Atomic Committed Configurations

Table 3: Feature History Table

Feature Name	Release Information	Description
Prevent Partial Pseudo-Atomic Committed Configurations	Release 7.10.1	You can now prevent the partially-committed configurations on the router and thus ensure the system database and OpenConfig datastore stay in sync. This feature changes how the internal rollback error is handled when a pseudo-atomic commit fails. In such cases, the system database always rolls back the configuration in its datastore thereby ensuring that there is no partially-committed configuration. If there is still inconsistency, the system displays error messages to notify you of various internal rollback failure scenarios based on which you must take rectification action to re-synchronize the data.

Existing Pseudo-Atomic Commit Behavior

The default behavior in pseudo-atomic commit is that all changes must succeed for the entire commit operation to succeed. If any errors are found, none of the configuration changes take effect.

Thus if an error occurs in one or more of the configurations in a commit, other configurations which were already successfully processed as part of the commit process are reverted. An internal rollback mechanism takes effect and reverts the already successful configurations to their original state.

Occasionally, the internal rollback may fail, that is, the verifier process rejects the rollback configuration. To stay in-sync with the verifier, the system database also does not rollback the configuration. This leads to commit of the failed-to-rollback configurations and results with system having partial committed configuration.

You can view the partial configuration with **show config commit changes [commit_id]** and take necessary action to keep the system database in-sync with verifiers.

Enhanced Pseudo-Atomic Commit Behavior

From IOS XR Software Release 7.10.1 onwards, for XR OpenConfig support, the running configurations in OpenConfig datastore can only be updated atomically. When the pseudo-atomic commit fails and the verifier rejects a rollback, OpenConfig datastore and system database would be out of sync in the existing pseudo-atomic commit behavior. The OpenConfig datastore would contain no changes from the commit, whereas the system database would contain configurations that failed to be rolled back.

The enhanced pseudo-atomic commit feature changes the way the internal rollback error is handled after a pseudo-atomic commit fails. This ensures the system database and OpenConfig datastore database stay in sync.

When the verifier process fails the configuration during an internal rollback, system database displays an `ios` error message to warn about the verifier error. You must take rectification action and re-synchronize the verifier and the system database. A failure to notice the error message or failure to restart the verifier process results in an inconsistent or deceptive operation of the system. After a while, the rollback error would become untraceable and could manifest into more problems.

Following are the scenarios with examples, where the internal rollback error appears when a pseudo-atomic commit fails:

- When the verifier process rejects the configuration during an internal rollback, system database displays an error message and continues to update system database and instruct the verifier to apply the configuration.

```
%MGBL_VERIFIER-4-COMMIT_ROLLBACK_REJECTED
```

Example shows the name of the process which rejects the internal rollback:

```
%MGBL-VERIFIER-4-COMMIT_ROLLBACK_REJECTED : verify_process incorrectly rejected rollback
 of a failed commit to a previously accepted state.  The rollback change has been made
 anyway. (/cfg/gl/test/item1, 0x40828400)
```

- When there is a timeout in the verify event (system database does not receive response from verifier within 300 seconds), then system database displays an `ios` message to warn you about the verifier timeout error and continue to update system database and instruct the verifier to apply the configuration.

```
%MGBL_VERIFIER-4-COMMIT_ROLLBACK_TIMEOUT
```

Example shows the name of the process which timeout for the internal rollback:

```
%MGBL-VERIFIER-3-COMMIT_ROLLBACK_TIMEOUT : verify_process (jid 68368, 0/0/CPU0) took
too long to verify the rollback of a failed commit
(cfg/if/act/GigabitEthernet0_0_0_2/a/test/item3). The rollback change has been made
anyway.
```

- When the verifier process fails to apply the internal rollback configuration or when the apply callback timeout, then the system database displays an `ios` message to warn you about the rollback failure and how to rectify the error by restarting the verifier process.

```
%MGBL_VERIFIER-3-COMMIT_ROLLBACK_FAILED
```

Example shows the name of the process which failed the internal rollback:

```
%MGBL-VERIFIER-3-COMMIT_ROLLBACK_FAILED : verify_process failed to apply the rollback
of a failed commit (/cfg/gl/test/item1, 0x40828400) and may no longer operate as
configured.  The process need to be restarted to rectify the error.
```

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:

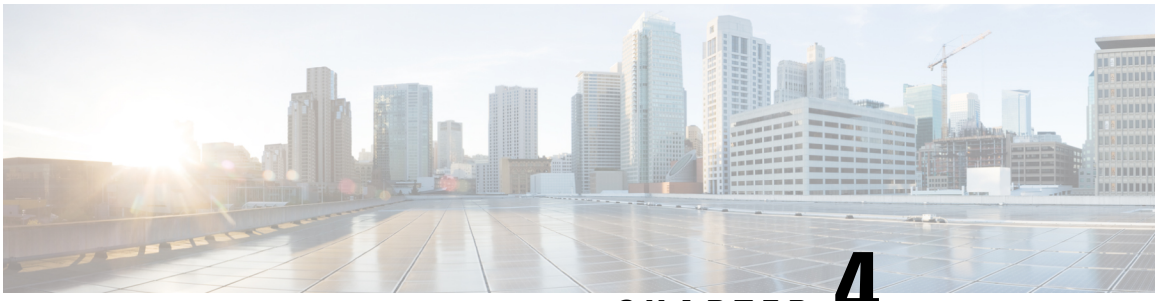
```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<delete xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">
    <name>harddisk:/netconf.txt</name>
  </delete>
</rpc>
```

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
```



CHAPTER 4

Use NETCONF Protocol to Define Network Operations with Data Models

Table 4: 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
```

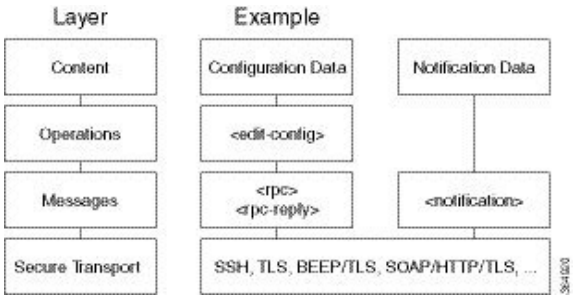
To enable the NETCONF SSH agent, use the following command:

```
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 26
- Retrieve Default Parameters Using with-defaults Capability, on page 30
- Retrieve Transaction ID for NSO Operations, on page 36
- Set Router Clock Using Data Model in a NETCONF Session, on page 38
- NETCONF Version 1.0 with YANG Support, on page 42

NETCONF Operations

NETCONF defines one or more configuration datastores and allows configuration operations on the datastores. A configuration datastore is a complete set of configuration data that is required to get a device from its initial default state into a desired operational state. The configuration datastore does not include state data or executive commands.

The base protocol includes the following NETCONF operations:

```
|  +--get-config
|  +--edit-Config
|     +--merge
|     +--replace
|     +--create
|     +--delete
|     +--remove
|     +--default-operations
|        +--merge
|        +--replace
|        +--none
|  +--get
|  +--lock
|  +--unLock
|  +--close-session
|  +--kill-session
```

These NETCONF operations are described in the following table:

NETCONF Operation	Description	Example
<get-config>	Retrieves all or part of a specified configuration from a named data store	Retrieve specific interface configuration details from running configuration using filter option <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <get-config> <source> <running/> </source> <filter> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"\> <interface-configuration> <active>act</active> <interface-name>TenGigE0/0/0/2/0</interface-name> </interface-configuration> </interface-configurations> </filter> </get-config> </rpc>

NETCONF Operation	Description	Example
<get>	Retrieves running configuration and device state information	Retrieve all acl configuration and device state information. Request: <get> <filter> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/> </filter> </get>
<edit-config>	Loads all or part of a specified configuration to the specified target configuration	Configure ACL configs using **Merge** operation <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target><candidate/></target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-cfg" xc:operation=”merge”> <accesses> <access> <access-list-name>aclv4-1</access-list-name> <access-list-entries> <access-list-entry> <sequence-number>10</sequence-number> <remark>GUEST</remark> </access-list-entry> <access-list-entry> <sequence-number>20</sequence-number> <grant>permit</grant> <source-network> <source-address>172.0.0.0</source-address> <source-wild-card-bits>0.0.255.255</source-wild-card-bits> </source-network> </access-list-entry> </access-list-entries> </access> </accesses> </ipv4-acl-and-prefix-list> </config> </edit-config> </rpc> Commit: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <commit/> </rpc>

NETCONF Operation	Description	Example

<lock>	Allows the client to lock the entire configuration datastore system of a device	Lock the running configuration.
		`Request:`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<lock>`
		`<target>`
		`<running/>`
		`</target>`
		`</lock>`
		`</rpc>`
		
		`Response :`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		
		`<ok/>`
		`</rpc-reply>`

<Unlock>	Releases a previously locked configuration.
	An <unlock> operation will not succeed if either of the following conditions is true:
	• The specified lock is not currently active.
	• The session issuing the <unlock> operation is not the same session that obtained the lock.
		Lock and unlock the running configuration from the same session.
		`Request:`
		`rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<unlock>`
		`<target>`
		`<running/>`
		`</target>`
		`</unlock>`
		`</rpc>`
		
		`Response -`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

<close-session>	Closes the session. The server releases any locks and resources associated with the session and closes any associated connections.	Close a NETCONF session.
		`Request :`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<close-session/>`
		`</rpc>`
		
		`Response:`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

NETCONF Operation	Description	Example

<kill-session>	Terminates operations currently in process, releases locks and resources associated with the session, and close any associated connections.	Terminate a session if the ID is other session ID.
		`Request:`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<kill-session>`
		`<session-id>4</session-id>`
		`</kill-session>`
		`</rpc>`
		
		`Response:`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

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 5: 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	XR7
cfgmgr	Yes
sysadmin	No
cfgmgr-aug	No
Leaf Data	cfgmgr
Check synchronization (NSO functionality from release 7.4.1 and later)	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

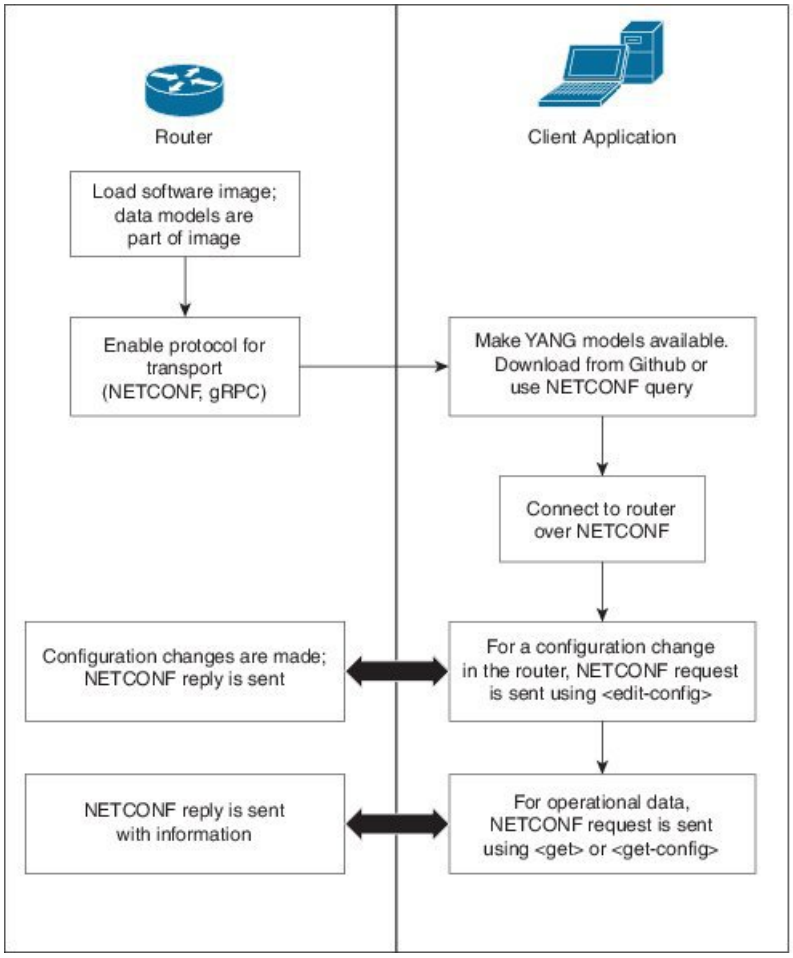
The process for using data models involves:

- 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

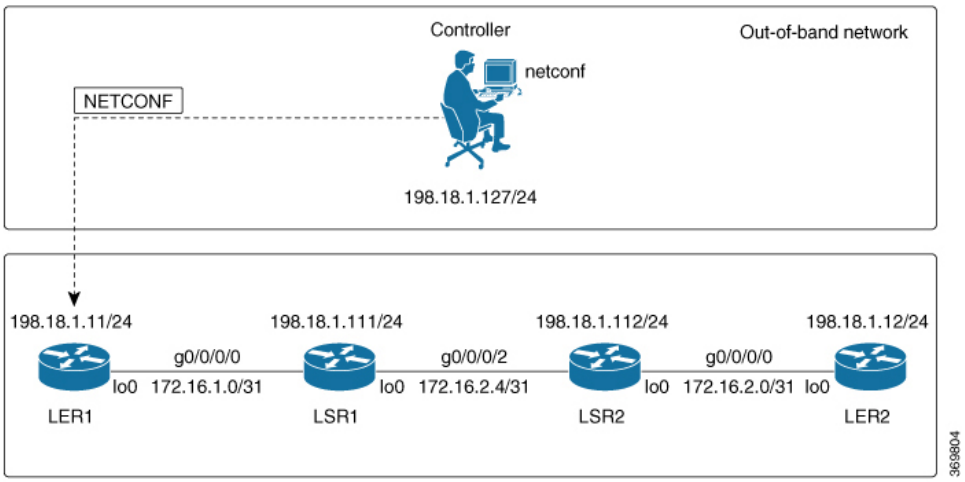


In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

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.

NETCONF Version 1.0 with YANG Support

Table 6: Feature History Table

Feature Name	Release Information	Feature Description
NETCONF Version 1.0 with YANG Support	Release 24.2.11	You can now monitor and manage a larger number of network devices, ensuring comprehensive oversight and control over your network infrastructure with NETCONF-YANG version 1.0. This enhancement is possible because our system has increased the support for NETCONF YANG sessions from 50 to 128.

NETCONF Version 1.0 and YANG Integration

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

To connect to netconf-yang agent with netconf version 1.0, it's necessary to configure the **netconf-yang agent netconf version1.0** command, otherwise a client with netconf 1.0 connects to a netconf-xml agent.

Prerequisites

- Install the software package `k9sec pie` and `mgbl pie` on the router.
- Generate the crypto keys.

Note

- NETCONF agent TTY becomes mutually exclusive with **netconf-yang agent netconf1.0 [only] [streaming-disabled]** command.
  - **only**—Netconf 1.1 is disabled in which the client supports both 1.0 and 1.1
  - **streaming-disabled**—snetconf-yang 1.0 agent disables sending the large streaming data in multiple chunks.
- To connect to netconf-yang agent with NETCONF1.0, use a NETCONF client that is compatible with NETCONF 1.0 standard (RFC-4741 and RFC-4742).
- If a NETCONF client supports both 1.0 and 1.1 versions, the NETCONF client connects with 1.1 as per the RFC-4741 and RFC-4742.
- To connect with 1.0, enable netconf-yang 1.0 with option `only ::: {custom-style="Block Label"} netconf-yang agent netconf1.0 only :::`
- By default, the netconf-yang 1.0 agent sends a large response data in multiple chunks.
- If a NETCONF client isn't able to handle receiving multiple data chunks, it can be turned off by configuring netconf-yang 1.0 with the option `streaming-disabled ::: {custom-style="Block Label"} netconf-yang agent netconf1.0streaming-disabled :::`

Configure NETCONF-YANG Version 1.0

Configuration Example

To configure NETCONF_YANG Version 1.0 session limit, use the **netconf-yang agent session limit** command.

Note To set a **netconf-yang agent session limit** on a NETCONF-YANG agent, first enable **netconf-yang agent netconf1.0** on the agent.

```
Router# config
Router(config)# netconf-yang agent netconf1.0
Router(config)# netconf-yang agent session limit 10

/*limit value sets the maximum count for concurrent netconf-yang sessions. The range is
from 1 to 128.*/
Router# end
```

Verification

To verity the NETCONF YANG statistics and NETCONF YANG clients, use the **do show netconf-yang statistics** and **do show netconf-yang clients** commands.

```
/*Verify Configuration Using Statistics*/
Router# do show netconf-yang statistics

Summary statistics        requests|             total time|   min time per request|   max
time per request|   avg time per request|

other                              0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

close-session                      4|       0h  0m  0s   3ms|       0h  0m  0s   0ms|
 0h  0m  0s   1ms|       0h  0m  0s   0ms|

kill-session                       0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

get-schema                         0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

get                                0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s

get-config                         1|       0h  0m  0s   1ms|       0h  0m  0s   1ms|
 0h  0m  0s   1ms|       0h  0m  0s   1ms|

edit-config                        3|       0h  0m  0s   2ms|       0h  0m  0s   0ms|
 0h  0m  0s   1ms|       0h  0m  0s   0ms|

commit                             0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

cancel-commit                      0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

lock                               0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

unlock                             0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

discard-changes                    0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|

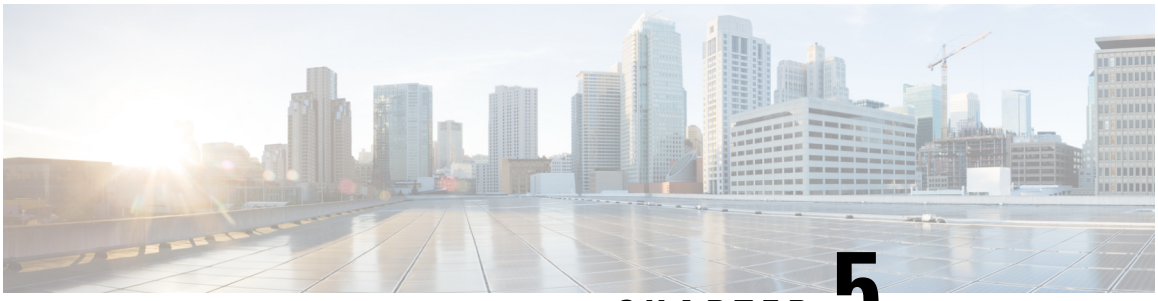
validate                           0|       0h  0m  0s   0ms|       0h  0m  0s   0ms|
 0h  0m  0s   0ms|       0h  0m  0s   0ms|


/*Verify Configuration Using Clients/*

Router# do show netconf-yang clients

client session ID|   NC version|    client connect time|        last OP time|        last
OP type|    <lock>|

22969|                      1.0|         0d  0h  0m  2s|            11:11:24|
close-session|        No|
```



CHAPTER 5

Use gRPC Protocol to Define Network Operations with Data Models

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

gRPC is an open-source RPC framework. It is based on Protocol Buffers (Protobuf), which is an open source binary serialization protocol. gRPC provides a flexible, efficient, automated mechanism for serializing structured data, like XML, but is smaller and simpler to use. You define the structure using protocol buffer message types in `.proto` files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs.

gRPC encodes requests and responses in binary. gRPC is extensible to other content types along with Protobuf. The Protobuf binary data object in gRPC is transported over HTTP/2.

gRPC supports distributed applications and services between a client and server. gRPC provides the infrastructure to build a device management service to exchange configuration and operational data between a client and a server. The structure of the data is defined by YANG models.

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

Cisco gRPC IDL uses the protocol buffers interface definition language (IDL) to define service methods, and define parameters and return types as protocol buffer message types. The gRPC requests are encoded and sent to the router using JSON. Clients can invoke the RPC calls defined in the IDL to program the router.

The following example shows the syntax of the proto file for a gRPC configuration:

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

    rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};

    rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};

    rpc DeleteConfig(ConfigArgs) returns(ConfigReply) {};
```

```
    rpc ReplaceConfig(ConfigArgs) returns(ConfigReply) {};

    rpc CliConfig(CliConfigArgs) returns(CliConfigReply) {};

    rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

    rpc CommitReplace(CommitReplaceArgs) returns(CommitReplaceReply) {};
}
message ConfigGetArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

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

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

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

message ConfigArgs {
    int64 ReqId = 1;
    string yangjson = 2;

}

message ConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

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

message CliConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CommitReplaceArgs {
     int64 ReqId = 1;
     string cli = 2;
     string yangjson = 3;
}

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

Example for gRPCExec configuration:

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

}

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

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

Example for OpenConfiggRPC configuration:

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

message GetModelsInput {
     uint64  requestId   = 1;
     string  name        = 2;
    string  namespace   = 3;
    string  version     = 4;
    enum MODLE_REQUEST_TYPE {
         SUMMARY = 0;
         DETAIL  = 1;
     }
     MODLE_REQUEST_TYPE requestType = 5;
}

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

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

gRPC Operations

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)

rpc GetConfig
{
"Cisco-IOS-XR-cdp-cfg:cdp": [
"cdp": "running-configuration"
]
}
 	
rpc GetConfig
{
"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": [
"lldp": "running-configuration"
]
}
	{
	"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
	}
	}

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 7: 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*.

Authenticate gRPC Services

Note Typically, gRPC clients include the username and password in the gRPC metadata fields.

Use any one of the following configuration type to authenticate any gRPC service.

- **Metadata with TLS**

```
Router#config
Router(config)#grpc
Router(config-grpc)#commit
```

- **Metadata without TLS**

```
Router#config
Router(config)#grpc
```

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

- **Metadata with Mutual TLS**

```
Router#config
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#commit
```

- **Certificate based Authentication**

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

SPIFFE ID-Based Authentication and Authorization Services for gRPC Services

Table 8: Feature History Table

Feature Name	Release Information	Description
SPIFFE ID-Based Authentication and Authorization Services for gRPC Services	Release 24.2.11	You can now securely manage service identities for workloads that communicate over gRPC. This capability is critical for environments such as distributed systems, where workloads move across different platforms. This security measure is feasible because workloads can use the Secure Production Identity Framework for Everyone (SPIFFE) ID and SPIFFE Verifiable Identity Document (SVID) to encrypt and authenticate gRPC traffic. This feature introduces the following changes: CLI: • **aaa map-to username** Yang Data Models: • New XPaths for `Cisco-IOS-XR-um-aaa-task-user-cfg.yang` • New XPaths for `Cisco-IOS-XR-aaa-locald-cfg.yang` (see GitHub, YANG Data Models Navigator)

The SPIFFE standard specifies a framework that can bootstrap and issue identities to services across diverse environments and organizational boundaries. SPIFFE assigns a unique identity to each workload with a SPIFFE ID and securely encapsulates it within a SPIFFE Verifiable Identity Document (SVID). The SVID, which is short-lived, corresponds exclusively to its SPIFFE ID and can be encoded either as an X.509 certificate or as a JSON Web Token (JWT). This dual-format capability facilitates robust identity verification.

This feature provides a mechanism for mapping a SPIFFE ID to an XR user for authorization purposes. This feature enables Extensible Manageability Services (EMS) to use the SVID, which are certificates that essentially contain SPIFFE IDs, to perform the following operations:

- Authentication via mTLS
- AuthZ authorization using the SVID

The XR authorization occurs with the XR user which is mapped to the SPIFFE ID. Mapping the SPIFFE ID to a username is required for gRPC services to perform IOS XR authentication and authorization before executing any operations on the device. If the authz evaluation is successful then only the connection request is processed; otherwise, access is denied.

Workflow for SPIFFE ID-Based Authentication and Authorization for gRPC Services

The high-level workflow of SPIFFE ID-based authentication and authorization for gRPC services involves the following steps:

1. The EMS starts searching for the *spiffe-user-map.json* file at the location `/misc/config/grpc/gnsi/credentialz/spiffe-user-map.json`.
2. If the file exists, it is parsed, and the mapping is stored globally in the `aaa/auth package`.
3. If the file does not exist or parsing is unsuccessful, the mapping will be empty.
4. The EMS registers with the configuration manager to receive updates for the `aaa` configuration.
5. When processing requests in the Authentication interceptor, the spiffe-user mapping API checks for the SPIFFE ID mapping in the map created in step 2.
6. If the mapping exists, the API responds with the corresponding username.
7. If the mapping does not exist but the `aaa` configuration exists, the API responds with the configured username.
8. If neither the mapping nor the `aaa` configuration is present, the API responds with an empty string.
9. Upon a client connecting to the server, the server interceptor extracts the SPIFFE ID from the client's certificate and uses the mapping stored in the `aaa/auth package` to find the corresponding username.
10. The username identifies it and then includes the metadata into the context.
11. gRPC services that require XR Authorization will later verify the access rights for the username identified in the previous step when handling the request.
12. If the mapping is unsuccessful, the request is passed to the relevant service, such as gNMI, which then decides whether to grant or deny access based on its authorization requirements.

Authenticate and Authorize gRPC Service Requests Using the SPIFFE Standard

Before you begin

Before authenticating and authorizing gRPC service requests using the SPIFFE standard, ensure the following prerequisites are met:

- Enable mutual TLS authentication with the `tls-mutual` command.
- Enable certificate authentication with the `certificate-authentication` command to facilitate SPIFFE ID recognition. For more information, see Authenticate gRPC Services, on page 50.
- Configure the gNSI Authz policy by setting the principal to the SPIFFE-ID for service-level authorization (gNSI AuthZ).

After establishing the connection, the gRPC server extracts the SPIFFE ID from the client's certificate.

To authenticate and authorize gRPC service requests using the SPIFFE standard, follow these steps:

Step 1 Configure the username in the system.

Example:

```
Router#show running-config aaa
Thu Oct 12 11:43:15.771 UTC
username cisco
 group root-lr
 group cisco-support
 password 7 104D000A061843595F
!
```

Step 2 Map the SPIFFE ID to a username using the **aaa map-to username** command. This command assigns a default username to any SPIFFE ID.

```
Router(config)#aaa map-to username cisco spiffe-id any
Router(config)#commit
```

Note Each SPIFFE ID supports only one username.

Step 3 Evaluate the client's SPIFFE ID against the service-level authorization policy (gNSI AuthZ). For more information about gNSI authz policies, see gRPC Network Security Interface , on page 87.

Certificate Common-Name For Dial-in Using gRPC Protocol

Table 9: 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.

gRPC over UNIX Domain Sockets

Table 10: Feature History Table

Feature Name	Release Information	Description
gRPC Connections over UNIX domain sockets for Enhanced Security and Control	Release 7.5.1	This feature allows local containers and scripts on the router to establish gRPC connections over UNIX domain sockets. These sockets provide better inter-process communication eliminating the need to manage passwords for local communications. Configuring communication over UNIX domain sockets also gives you better control of permissions and security because UNIX file permissions come into force. This feature introduces the **grpc local-connection** command.

You can use local containers to establish gRPC connections via a TCP protocol where authentication using username and password is mandatory. This functionality is extended to establish gRPC connections over UNIX domain sockets, eliminating the need to manage password rotations for local communications.

When gRPC is configured on the router, the gRPC server starts and then registers services such as gRPC Network Management Interface and gRPC Network Operations Interface . After all the gRPC server registrations are complete, the listening socket is opened to listen to incoming gRPC connection requests. Currently, a TCP listen socket is created with the IP address, VRF, or gRPC listening port. With this feature, the gRPC server listens over UNIX domain sockets that must be accessible from within the container via a local connection by default. With the UNIX socket enabled, the server listens on both TCP and UNIX sockets. However, if disable the UNIX socket, the server listens only on the TCP socket. The socket file is located at /var/lib/docker/ems/grpc.sock directory.

The following process shows the configuration changes required to enable or disable gRPC over UNIX domain sockets.

Step 1 Configure the gRPC server.

Example:

```
Router(config)#grpc
Router(config-grpc)#local-connection
Router(config-grpc)#commit
```

To disable the UNIX socket use the following command.

```
Router(config-grpc)#no local-connection
```

The gRPC server restarts after you enable or disable the UNIX socket. If you disable the socket, any active gRPC sessions are dropped and the gRPC data store is reset.

The scale of gRPC requests remains the same and is split between the TCP and Unix socket connections. The maximum session limit is 256, if you utilize the 256 sessions on Unix sockets, further connections on either TCP or UNIX sockets is rejected.

Step 2 Verify that the local-connection is successfully enabled.

Example:

```
Router#show grpc status
Thu Nov 25 16:51:30.382 UTC
*************************show gRPC status**********************
----------------------------------------------------------------
transport                       :     grpc
access-family                   :     tcp4
TLS                             :     enabled
trustpoint                      :
listening-port                  :     57400
local-connection                :     enabled
max-request-per-user            :     10
max-request-total               :     128
max-streams                     :     32
max-streams-per-user            :     32
vrf-socket-ns-path              :     global-vrf
min-client-keepalive-interval   :     300
________________________________________________________________
```

A gRPC client must dial into the socket to send connection requests.

The following is an example of a Go client connecting to UNIX socket:

```
const sockAddr =
"/var/lib/docker/ems/grpc.sock"
...
func UnixConnect(addr string, t time.Duration) (net.Conn, error) {
    unix_addr, err := net.ResolveUnixAddr("unix", sockAddr)
    conn, err := net.DialUnix("unix", nil, unix_addr)
    return conn, err
}


func main() {
...
    opts = append(opts, grpc.WithTimeout(time.Second*time.Duration(*operTimeout)))
    opts = append(opts, grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(math.MaxInt32)))
    ...
    opts = append(opts, grpc.WithDialer(UnixConnect))
    conn, err := grpc.Dial(sockAddr, opts...)
    ...
}
```

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.

gNMI Operations

gNMI Operation	Supported Release	Description	Additional Details
Capabilities	Release 7.0.1	Retrieves the metadata of the network device.	—
Get	Release 7.0.1	Retrieve state data, configuration, and operational information from a network device	—
Set	Release 7.0.1	You can modify the state of a network device such as router's configuration, replace router's entire configuration sections, or delete specific parts of the configuration using the **Set** operation.	—
Subscribe	Release 24.2.1	Subscribes to a stream of updates for specific paths within the device's data model.	Stream Telemetry Data for LLDP Statistics

gNMI Wildcard in Schema Path

Table 11: Feature History Table

Feature Name	Release Information	Description
Use gNMI Get Request With Wildcard Key to Retrieve Data	Release 7.5.2	You use a gRPC Network Management Interface (gNMI) `Get` request with wildcard key to retrieve the configuration and operational data of all the elements in the data model schema paths. In earlier releases, you had to specify the correct key to retrieve data. The router returned a JSON error message if the key wasn't specified in a list node. For more information about using wildcard search in gNMI requests, see the Github repository.

gNMI protocol supports wildcards to indicate all elements at a given subtree in the schema. These wildcards are used for telemetry subscriptions or gNMI `Get` requests. The encoding of the path in gNMI uses a structured

format. This format consists of a set of elements such as the path name and keys. The keys are represented as string values, regardless of their type within the schema that describes the data. gNMI supports the following options to retrieve data using wildcard search:

- **Single-level wildcard:** The name of a path element is specified as an asterisk (*). The following sample shows a wildcard as the key name. This operation returns the description for all interfaces on a device.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
     key: "name"
      value: "*"
    }
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

- **Multi-level wildcard:** The name of the path element is specified as an ellipsis (…). The following example shows a wildcard search that returns all fields with a description available under `/interfaces` path.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "..."
  }
  elem {
    name: "description"
  }
}
```

Example: gNMI Get Request with Unique Path to a Leaf

The following is a sample `Get` request to fetch the operational state of `GigabitEthernet0/0/0/0` interface in particular.

```
path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
        elem: <
            name: "interfaces"
        >
        elem: <
            name: "interface-xr"
        >
        elem: <
            name: "interface"
            key: <
                key: "interface-name"
                value: "\"GigabitEthernet0/0/0/0\""
            >
        >
        elem: <
```

```
            name: "state"
        >
>
type: OPERATIONAL
encoding: JSON_IETF
```

The following is a sample `Get` response:

```
notification: <
  timestamp: 1597974202517298341
  update: <
    path: <
      origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
       elem: <
          name: "interfaces"
       >
       elem: <
          name: "interface-xr"
       >
       elem: <
          name: "interface"
          key: <
            key: "interface-name"
            value: "\"GigabitEthernet0/0/0/0\""
           >
        >
        elem: <
            name: "state"
           >
       >
       val: <
          json_ietf_val: im-state-admin-down
       >
    >
>
error: <
>
```

Example: gNMI Get Request Without a Key Specified in the Schema Path

The following is a sample `Get` request to fetch the operational state of all interfaces.

```
path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
        elem: <
            name: "interfaces"
        >
        elem: <
            name: "interface-xr"
        >
        elem: <
            name: "interface"
        >
        elem: <
            name: "state"
        >
>
type: OPERATIONAL
encoding: JSON_IETF
```

The following is a sample `Get` response:

```
path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
```

```
        elem: <
            name: "interfaces"
        >
        elem: <
            name: "interface-xr"
        >
        elem: <
            name: "interface"
        >
        elem: <
            name: "state"
        >
>
type: OPERATIONAL
encoding: JSON_IETF
  notification: <
  timestamp: 1597974202517298341
  update: <
    path: <
      origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface-xr"
      >
      elem: <
        name: "interface"
        key: <
          key: "interface-name"
          value: "\"GigabitEthernet0/0/0/0\""
        >
      >
      elem: <
        name: "state"
      >
    >
    val: <
      json_ietf_val: im-state-admin-down
    >
  >
  update: <
    path: <
      origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface-xr"
      >
      elem: <
        name: "interface"
        key: <
          key: "interface-name"
          value: "\"GigabitEthernet0/0/0/1\""
        >
      >
      elem: <
        name: "state"
      >
    >
    val: <
      json_ietf_val: im-state-admin-down
    >
```

```
>
update: <
  path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
    elem: <
      name: "interfaces"
    >
    elem: <
      name: "interface-xr"
    >
    elem: <
      name: "interface"
      key: <
        key: "interface-name"
        value: "\"GigabitEthernet0/0/0/2\""
      >
    >
    elem: <
      name: "state"
    >
  >
  val: <
    json_ietf_val: im-state-admin-down
  >
>
update: <
  path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
    elem: <
      name: "interfaces"
    >
    elem: <
      name: "interface-xr"
    >
    elem: <
      name: "interface"
      key: <
        key: "interface-name"
        value: "\"MgmtEth0/RP0/CPU0/0\""
      >
    >
    elem: <
      name: "state"
    >
  >
  val: <
    json_ietf_val: im-state-admin-down
  >
>
```

gNMI Bundling of Telemetry Updates

Table 12: Feature History Table

Feature Name	Release Information	Description
gNMI Bundling Size Enhancement	Release 7.8.1	With gRPC Network Management Interface (gNMI) bundling, the router internally bundles multiple gNMI `Update` messages meant for the same client into a single gNMI `Notification` message and sends it to the client over the interface. You can now optimize the interface bandwidth utilization by accommodating more gNMI updates in a single notification message to the client. We have now increased the gNMI bundling size from 32768 to 65536 bytes, and enabled gNMI bundling size configuration through Cisco native data model. Prior releases allowed only a maximum bundling size of 32768 bytes, and you could configure only through CLI. The feature introduces new XPaths to the `Cisco-IOS-XR-telemetry-model-driven-cfg.yang` Cisco native data model to configure gNMI bundling size. To view the specification of gNMI bundling, see Github repository.

To send fewer number of bytes over the gNMI interface, multiple gNMI `Update` messages pertained to the same client are bundled and sent to the client to achieve optimized bandwidth utilization.

The router internally bundles multiple gNMI `Update` messages in a single gNMI `Notification` message of gNMI `SubscribeResponse` message. Cisco IOS XR software Release 7.8.1 supports gNMI bundling size up to 65536 bytes.

Router bundles multiple instances of the same client. For example, a router bundles interfaces `MgmtEth0/RP0/CPU0/0`, `FourHundredGigE0/0/0/0`, `FourHundredGigE0/0/0/1`, and so on, of the following path.

- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`

Router does not bundle messages of different client in a single gNMI `Notification` message. For example,

- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`
- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/protocols`

  Data under the container of the client path cannot be split into different bundles.

The gNMI `Notification` message contains a timestamp at which an event occurred or a sample is taken. The bundling process assigns a single timestamp for all bundled `Update` values. The notification timestamp is the first message of the bundle.

Note

- ON-CHANGE subscription mode does not support gNMI bundling.
- Router does not enforce bundling size in the following scenarios:
  - At the end of (N-1) message processing, if the notification message size is less than the configured bundling size, router allows one extra instance which could result in exceeding the bundling size.
  - Data of a single instance exceeding the bundling size.
- The XPath: `network-instances/network-instance/afts` does not support bundling.

Configure gNMI Bundling Size

gNMI bundling is disabled by default and the default bundling size is 32,768 bytes. gNMI bundling size ranges from 1024 to 65536 bytes. Prior to Cisco IOS XR software Release 7.8.1 the range was 1024 to 32768 bytes. You can enable gNMI bundling to all gNMI subscribe sessions and specify the bundling size.

Configuration Example

This example shows how to enable gNMI bundling and configure bundling size.

```
Router# configure
Router(config)# telemetry model-driven
Router(config-model-driven)# gnmi
Router(config-gnmi)# bundling
Router(config-gnmi-bdl)# size 2000
Router(config-gnmi-bdl)# commit
```

Running configuration

This example shows the running configuration of gNMI bundle.

```
Router# show running-config
telemetry model-driven
 gnmi
  bundling
   size 2000
  !
 !
!
```

Replace Router Configuration at Sub-tree Level Using gNMI

Table 13: Feature History Table

Feature Name	Release Information	Description
Replace Router Configuration at Sub-tree Level Using gNMI	Release 7.8.1	Using the gNMI `SetRequest` message, you can replace the router's existing configuration with a new set of configurations at the subtree level within the same model. Earlier you could replace router configurations at the data tree root level. To view the specification of gNMI replace, see Github repository.

The gNMI replace feature replaces the existing configuration on the router with the new configuration using a `SetRequest` RPC message. It allows you to specify a `path` (a structured format for path elements, and any associated key values) as the root prompt to perform a `replace` operation. Cisco IOS XR software Release 7.8.1 supports subtree-level replace operation. Prior to this release replace operation was performed at datatree-level.

Replace operation either includes all the path elements which are defined under the root or only few of them. If the omitted path elements are configured with default values, they are reverted to its default values during the replace operation. If the omitted path elements are not configured with default values, they are deleted from the data tree during the replace operation, and returned to its original unconfigured state. Consider the following example:

In the following data tree schema, `b` has a default value of `true` and `c` has no default value. Both `b` and `c` are set as `False`.

```
root +
     |
     + a --+
     |     |
     |     +-- b
     |     |
     |     +-- c
     |
     |
     + d --+
           +-- e
           |
           +-- f
```

When a `replace` operation is performed with `e` and `f` as set, and all other elements are omitted, `b` is reverted to its default setting true, and `c` is deleted from the tree, and returned to its original unconfigured state.

Following example shows the `SetRequest` and `SetResponse` of gNMI replace operation.

gNMI Replace Example

This example shows the gNMI replace request and response messages.

```
Request Message:
replace: <
  path: <
    elem: <
      name: "system"
    >
```

```
    elem: <
      name: "config"
    >
    elem: <
      name: "hostname"
    >
  >
  val: <
    json_ietf_val: "\"testing123\""
  >
>

Response Message:
  path: <
    elem: <
      name: "system"
    >
    elem: <
      name: "config"
    >
    elem: <
      name: "hostname"
    >
  >
  op: REPLACE
>
message: <
>
timestamp: 1662873319202107537
```

gNMI Union Replace Operation

Table 14: Feature History Table

Feature Name	Release Information	Description
gNMI Union Replace Operation	Release 24.2.11	You can now update your router's entire configuration in one go to ensure that the actual settings of your network operating system align with the intended setup. The update includes OpenConfig (OC), Native YANG (NY), and CLI configurations and is done using the gRPC Network Management Interface (gNMI). The update is possible with the gNMI union-replace operation in a `gNMI SetRequest` RPC message which supports mixing of the configuration schemas. The supported schema combinations are: • OpenConfig (OC) and CLI • OC and native YANG (NY) To view the specification of gNMI union-replace, see the Github repository.

Routers can be configured using different schemas including native YANG (NY) models, the command-line interface (CLI), or OpenConfig (OC) YANG models. You can now update your router's entire configuration in one go to ensure that the actual settings of your network operating system align with the intended setup.

The router update can be done by merging these different schemas and directly replace the existing router settings using the gNMI union-replace operation.

gNMI Union-Replace Operation-Supported Schema Combinations

gNMI union-replace operation in a `gNMI SetRequest` RPC message supports the following two schema combinations:

- OC and CLI
- OC and NY

gNMI union-replace operation Guidelines and Limitations

Using gNMI when a client sends the `gNMI SetRequest` RPC message with union-replace operations to a target router:

- The state of the target router must not be changed until all the changes have been accepted successfully.
- If a particular path-value is specified in the gNMI request, the value replaces the current value in the target router.
- If a particular path-value isn't specified in the gNMI request and the path doesn't have a default value in the corresponding schema, it's deleted.
- If a path-value isn't specified in the gNMI request and the path does have a default value, the default value is applied on the target router.
- A `gNMI SetRequest` RPC message containing union_replace operations must not contain delete, replace, and update operations.

The origin field in the path message of a gNMI union-replace operation is set to one of the following:

- **openconfig**: Path and content are part of OC YANG models.
- **cisco_native**: Path and content are part of Cisco's network operating system YANG models.
- **cisco_cli**: This origin represents an ASCII text or CLI configuration defined as command-line interface (CLI) text.

If the origin field is unspecified, the origin value is set to OpenConfig.

gNMI Union Replace Operation Examples

The following schema combination examples show the union_replace operation in the `gNMI SetRequest` RPC message:

- OC and CLI Origin, on page 67
- OC and NY Origin, on page 69

OC and CLI Origin

gNMI union_replace operation in `gNMI SetRequest` RPC message with OC and CLI origin schema combination example is as follows:

```
union_replace: {
  path: {
    origin: "cisco_cli"
  }
  val: {
    ascii_val: "hostname myhost"
  }
}
union_replace: {
  path:  {
    elem:  {
      name:  "interfaces"
    }
    elem:  {
      name:  "interface"
      key: {
          key: "name"
          value: "FourHundredGigE0/0/0/0"
      }
    }
    elem:  {
      name:  "config"
    }
    elem:  {
      name: "description"
    }
  }
  val:  {
    json_ietf_val:  "\"true\""
  }
}
```

Replacement Sequence for the OC and CLI Origin Schema Combination

The configurations from both the schemas are merged and the merged configuration replaces the router's existing configuration.

Note If the CLI and OC configuration values overlap, the CLI configuration takes higher precedence and overwrites the value set by OC.

Guidelines for OC and CLI Origin

Ensure that you don't use a union-replace operation with an empty path under OC or CLI origins. Doing so removes all the content of the respective origin on the target router.

A union-replace operation with OC and CLI schema combination containing bootz configuration, the processing order of the configuration application on the target router is as follows: OC->CLI->bootz.

OC and NY Origin

A gNMI union_replace operation in the `gNMI SetRequest` RPC message with OC and NY origin schema combination example is as follows.

```
union_replace: {
  path: {
    origin: "cisco_native"
    elem: {
      name: "Cisco-IOS-XR-shellutil-cfg:host-names"
    }
    elem: {
      name: "host-name"
    }
  }
  val: {
    json_ietf_val: "\"abc\""
  }
}
union_replace: {
  path:  {
    elem:  {
      name:  "interfaces"
    }
```

```
    elem:  {
      name:  "interface"
      key: {
          key: "name"
                 value: "FourHundredGigE0/0/0/0"
      }
    }
    elem:  {
      name:  "config"
    }
    elem:  {
      name: "description"
    }
  }
  val:  {
    json_ietf_val:  "\"true\""
  }
}
```

Guidelines for OC and NY Origin

The configurations from both the schemas are merged and the merged configuration replaces the router's existing configuration.

If the OC and NY schema configuration values overlap, the NY configuration takes higher precedence and overwrites the value set by OC.

If an OC and NY union-replace requests explicitly set configuration items that are overlapping, the RPC doesn't return INVALID_ARGUMENT.

RPC Error Scenarios

The RPC message returns INVALID_ARGUMENT if:

- One of the origins from the supported schema combinations is missing or if the union_replace operation has no specified path value for one of the origins.
- Union-replace operations for all three origins (“cisco_native”, “cisco_cli”, and “openconfig”) are present in the `gNMI SetRequest` RPC message.
- A `gNMI SetRequest` RPC message with union_replace operations contain delete, replace, or update operations.

gNMI XPath-Based Authorization

Table 15: Feature History Table

Feature Name	Release Information	Description
gNMI XPath-Based Authorization	Release 24.2.11	We've introduced gNMI authorization through the gNSI pathz policy which is adding authorization of a user or a group to access a specified YANG XPath through gNMI. The policy configurations can be done on the router either when the router boots up or dynamically when the router is up and running. When a user or a group sends a `gNMI SetRequest` message using a certain XPath, the system validates the request against the permissions specified in the policies associated with that user or the group. To view the specification of gNSI for the OpenConfig XPath-based Authorization, see the Github repository. The feature introduces these changes: **CLI:** • show gnsi path authorization policy • show gnsi path authorization counters • show gnsi trace pathz • show gnsi path authorization statistics • show tech-support gnsi • clear gnsi path authorization counters

How gNSI pathz Policy Works

Upon receiving a `gNMI SetRequest` message for a configuration change, the router applies an XPath-based pathz policy to determine the request's authorization. The pathz policy originates from a gNSI RPC within the router. The policy configurations can be established during the router's boot process or dynamically adjusted while the router is operational.

The router securely receives the initial pathz policy either through Secure Zero Touch Provisioning (sZTP) or a secure bootstrapping protocol like bootz when booting up. The policy includes the user or group name and a list of rules defining XPaths and their associated access permissions. The policy is enforced before processing any gNMI requests.

Authorization by the gNSI pathz policy is granted or denied based on user or group credentials, permitting or declining the `gNMI SetRequest` accordingly.

gNMI Authorization Using gNSI pathz Policy

Starting from Release 24.2.11, you can perform gNMI XPath-based authorization using gNSI pathz policies.

The gnsi-pathz YANG model defines the following counters and timestamps for each configured rule READ, WRITE, PERMIT, and DENY.

- access-rejects: 64-bit
- last-access-reject: timestamp
- access-accepts: 64-bit
- last-access-accept: timestamp

The counters get incremented per accepted or rejected XPath (Example, per gNMI request).

Define Authorization Policy for a gNSI Pathz

The authorization policy for gNSI Pathz consists of three components.

Table 16: Authorization Policy Components

Authorization Policy Component	Details
Users	Individuals named in rules or group definitions.
Groups of users	A group of users in the administrative domain, such as operators or administrators. • The matching policy gives precedence to a specific user over a group. • Match rules enable authorization against either a user or a group, but not both simultaneously.
Policy rules	Each rule defines a single authorization policy. • Authorization (how the policy is defined) is performed for a specific user in a predefined group of users on a specific gNMI path and a specific access methodology (example: READ or WRITE). • The wildcard character (*): • Replaces the missing keys in keyed path elements. Absence of keys implies a wildcard by default. • Masks all the values entirely, it doesn't permit partial value masking (Example: /this/is/a/keyed[name=Ethernet1/*/3]/things is invalid).

How Authorization Policy Matching Rules Work

Policy Matching Rule	Description
Multiple rules	The authorization process evaluates the rule with the longest match when granting access, rather than defaulting to the first rule encountered.

Policy Matching Rule	Description
A defined KEY and wildcard in a keyed path	The defined KEY in the keyed path is preferred over the wildcard. For example, the router prefers /a/b[key=FOO]/c/d over /a/b[key=*]/c/d due to its more precise key match.
A user-specific rule and a corresponding group rule for the same user	The rule that corresponds to a specific user is prioritized over the one that matches with a user's group.
Permission mode	A mode that matches with the request (READ or WRITE) is considered.
DENY or PERMIT	DENY takes priority over PERMIT when other conditions are equal, and multiple matching rules are present.

Policy evaluation results with a single best match rule for the provided {user, path, or mode}. If multiple best matches emerge, an error is logged, and the evaluation fails.

If no matching rule is found, an implicit DENY is applied and detailed in a log entry.

The authorization evaluation process results in a PERMIT or DENY decision, along with the version of the policy and the identifier of the rule applied.

Scenario for Authorization Policy Rules

Rule	User	Group	Path	Action	Mode
1	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	READ
2	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
3	Bob	—	/interfaces/interface[FourHundredGigE1/1/1/1]	DENY	WRITE
4	—	Admin	/interfaces/interface[*]	PERMIT	WRITE
5	Bob	—	/interfaces	PERMIT	READ
6	—	Admin	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
7	Jim	—	/interfaces/interface[FourHundredGigE0/0/0/0]	DENY	WRITE

For user Bob, the following authorization rules apply:

- READ or WRITE (gNMI request) access to the XPath /interfaces/interface[FourHundredGigE0/0/0/0] is granted under rules 1 and 2.
- READ access to the XPath /interfaces/interface[FourHundredGigE1/1/1/1] is granted under rule 5 due to the longest match criterion, which specifies READ mode. WRITE access to this path is denied by rule 3.
- WRITE access to the XPath /interfaces/interface[FourHundredGigE2/2/2/2] is granted being a member of the Admins group as specified by rule 4. Without the Admin membership, access is denied by the default deny all rule.

- READ access to the XPath /interfaces/interface[FourHundredGigE2/2/2/2] is granted under rule 5, independent of group affiliation.

For user Jim, the following authorization rule applies:

- Access to the XPath /interfaces/interface[FourHundredGigE0/0/0/0] is controlled by a policy that favors personal user permissions over group permissions. As a result, although the admins group is allowed access, Jim is individually denied access because the policy emphasizes user-specific rules.

gNSI Pathz Authorization Policy Configuration

To set a gNSI pathz authorization policy, you can perform either of the following methods:

- Load gNSI Pathz Policies at Boot-time, on page 74
- Rotate, Finalize, and Get the gNSI Pathz Policy, on page 74

Load gNSI Pathz Policies at Boot-time

To load gNSI pathz policies at boot-time into the router, you can use either sZTP or bootstrapping.

For details on loading gNSI pathz policy through sZTP, refer to *Secure Zero Touch Provisioning* section of *Cisco IOS XR Setup and Upgrade Guide for Cisco 8000 Series Routers* guide.

Rotate, Finalize, and Get the gNSI Pathz Policy

When the router is up and running, you can rotate (update), finalize (commit), and get (read) the gNSI pathz policy using the gNSI pathz gRPC operations. To view the specification of gNSI pathz policy rotation, see the Github repository.

gNSI pathz supports the following policy instances:

- Active policy—Used for authorizing gNMI requests.
- Potential or candidate policy—Used to test a policy before rotation.

Rules for Authorization Policy Rotation

- The node holds on to the candidate policy indefinitely until either:
  - The candidate is committed or again rotated, or
  - The RPC session is closed (this event removes the candidate instance).
- A single policy rotation RPC can be active at any given time. Concurrent RPC requests for policy rotation is rejected with the gRPC error code `UNAVAILABLE`.
  - gNMI allows different encodings, including JSON. IOS XR applies the gNSI pathz policy based on each leaf of the flattened JSON model for authorizing the gNMI request.

Metrics of gNSI Authorization Rules

IOS-XR pathz supports the following statistics, counters, diagnostics, and trace data commands for the gNSI authorization rules:

- gNSI Pathz Policy and Statistics
- gNSI Path Authorization Counters
- gNSI Pathz Trace Data
- gNSI State Details

gNSI Path Authorization Counters

The gNSI path authorization counters show the counters for a given gRPC server-name for all XPaths, or the specified XPath. Providing the XPath and server-name is optional. To view the gNSI Path Authorization counters, use the **show gnsi path authorization counters** command.

- ```
  Router# show gnsi path authorization counters
  Mon Apr  1 08:05:46.297 UTC
  ----------------Pathz Counters Info--------------

  /system/config/hostname:
                                          Read                                Write
  Rejects :                                  0                                    0
     Last :                                N/A                                  N/A
  Accepts :                                  0                                    3
     Last :                                N/A    Mon, 01 Apr 2024 08:05:25 +0000
  Total path records received 1


  Router# show gnsi path authorization counters server-name 64.103.223.33
  Mon Apr  1 08:33:25.194 UTC
  ----------------Pathz Counters Info--------------

  /:
                                          Read                                Write
  Rejects :                                  0                                    2
     Last :                                N/A    Mon, 01 Apr 2024 08:32:37 +0000
  Accepts :                                  0                                    0
     Last :                                N/A                                  N/A

  /system/config/hostname:
                                          Read                                Write
  Rejects :                                  0                                    6
     Last :                                N/A    Mon, 01 Apr 2024 08:32:36 +0000
  Accepts :                                  0                                    0
     Last :                                N/A                                  N/A
  Total path records received 2
  Router#
  ```

  ```
  Router# show gnsi path authorization counters path /system/config/hostname
  Mon Apr  1 08:32:46.468 UTC
  ----------------Pathz Counters Info--------------

  /system/config/hostname:
                                          Read                                Write
  Rejects :                                  0                                    6
     Last :                                N/A    Mon, 01 Apr 2024 08:32:36 +0000
  Accepts :                                  0                                    0
     Last :                                N/A                                  N/A
  Total path records received 1
  Router#
  ```

- To clear the gNSI path authorization counters, use the **clear gnsi path authorization counters** command.

  ```
  Router# clear gnsi path authorization counters
  Router#
  ```
  ```

gNSI Pathz Policy and Statistics

To display the configured gNSI policy and statistics, use the are following commands:

- **show gnsi path authorization policy**—Shows the running gNSI path authorization policy.
- **show gnsi path authorization statistics**—Shows gNSI path authorization statistics.

```
Router# show gnsi path authorization policy
Mon Apr  1 04:29:37.905 UTC
version:"1" created_on:1711946719670313 policy:{rules:{user:"cafyauto"
path:{origin:"openconfig" elem:{name:"system"} elem:{name:"config"} elem:{name:"hostname"}}
 action:ACTION_PERMIT mode:MODE_WRITE}}
Router#
```

```
Router# show gnsi path authorization statistics
Mon Apr  1 04:29:23.259 UTC
----------------Pathz Info--------------
Engine:

  State:
    Active Policy:
      Version                 : 1
      Created On (UTC)        : Wed, 09 Dec 54251401 07:58:33 +0000
    Sandbox Policy:
      Version                 : N/A
      Created On (UTC)        : N/A
    Policy Rotation in Progress: False

  Stats:
    Rotations in Progress Count: 0
    Policy Rotations          : 0
    Policy Rotation Errors    : 0
    Policy Upload Requests    : 0
    Policy Upload Errors      : 0
    Policy Finalize           : 0
    Policy Finalize Errors    : 0
    Probe Requests            : 0
    Probe Errors              : 0
    Get Requests              : 0
    Get Errors                : 0
    Policy Unmarshall Errors  : 0
    Sandbox Policy Errors     : 0

  Counters:
    No Policy Auth Requests   : 0
    gNMI Path Leaves          : 0
    gNMI Authorizations       : 0
    gNMI Set Path Permit      : 0
    gNMI Set Path Deny        : 0
    gNMI Get Path Permit      : 0
    gNMI Get Path Deny        : 0

  Errors:
    Path To String            : 0
    Origin Type               : 0
    Bad Mode                  : 0
    Bad Action                : 0
    JSON Flatten              : 0
    String To Path            : 0
    Join Paths                : 0
    Nil Path                  : 0
    Nil SetRequest            : 0
    Empty User                : 0
```

```
    Probe Internal             : 0
    Path Counters:
      Increment                : 0
      Find                     : 0
      Clear                    : 0
      Walk                     : 0
```

gNSI Pathz Trace Data

To trace the configured gNSI policy, use the **show gnsi trace pathz** command.

```
Router# show gnsi trace pathz all
Mon Apr  1 04:31:26.689 UTC
61 wrapping entries (21760 possible, 512 allocated, 0 filtered, 61 total)
Apr  1 04:07:09.681 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(178) 'Trying to load policy'
'/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(173) 'Set Sandbox policy'
'1(54251382-02-18 11:34:58 +0000 UTC)'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(179) 'Set Policy from'
'/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(249) 'Pathz Policy Clearing
Counters' ' '
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code (79): 'Engine Initialized'
Apr  1 04:08:05.761 gnsi/pathz 0/RP0/CPU0 t11794 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:05.761 gnsi/pathz_err 0/RP0/CPU0 t11794 Pathz ERROR: Code (65): 'Nil Policy'
Apr  1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
Apr  1 04:08:05.791 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235) 'Upd/Rep Denied
 path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:05.808 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(234) 'Del Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:05.821 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(235) 'Upd/Rep Denied
 path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:07.348 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(235) 'Upd/Rep Denied
 path' 'cafyauto@/lldp/config/enabled,|1,1711946094752098'
Apr  1 04:08:08.205 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:08.205 gnsi/pathz_err 0/RP0/CPU0 t11383 Pathz ERROR: Code (65): 'Nil Policy'
Apr  1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
Apr  1 04:08:08.238 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235) 'Upd/Rep Denied
 path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:08.281 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(234) 'Del Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Router#
```

gNSI State Details

To collect diagnostic information of gNSI, use the **show tech-support gnsi** command.

```
Router# show tech-support gnsi
Mon Apr  1 06:55:51.482 UTC
++ Show tech start time: 2024-Apr-01.065551.UTC ++
Mon Apr  1 06:55:52 UTC 2024 Waiting for gathering to complete
...
Mon Apr  1 06:56:01 UTC 2024 Compressing show tech output
Show tech output available at Router#:
/harddisk:/showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551.UTC.tgz
++ Show tech end time: 2024-Apr-01.065601.UTC ++
```

show tech-support gnsi command places the collected diagnostic information in a file, example **Router#: /harddisk: /showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551**.

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.

gNOI RPCs

To send gNOI RPC requests, you need a client that implements the gNOI client interface for each RPC.

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

Table 17: Feature History Table

Feature Name	Release Information	Description
gNOI MPLS Proto	Release 7.5.4	The RPCs defined in the proto file can be used to perform Multiprotocol Label Switching (MPLS) operations on the router.
gNOI OS Proto	Release 7.9.1	The RPCs defined in the proto file can be used to install the software, activate the software version and verify that the installation is successful.
gNOI System Proto	Release 7.8.1	You can now avail the services of `CancelReboot` to terminate outstanding reboot request, and `KillProcess` RPCs to restart the process on device.

gNOI supports the following remote procedure calls (RPCs):

System RPCs

The RPCs are used to perform key operations at the system level such as upgrading the software, rebooting the device, and troubleshooting the network. The **system.proto** file is available in the Github repository.

RPC	Description
Reboot	Reboots the target. The router supports the following reboot options: • COLD = 1; Shutdown and restart OS and all hardware • POWERDOWN = 2; Halt and power down • HALT = 3; Halt • POWERUP = 7; Apply power
RebootStatus	Returns the status of the target reboot.
SetPackage	Places a software package including bootable images on the target device.
Ping	Pings the target device and streams the results of the ping operation.
Traceroute	Runs the traceroute command on the target device and streams the result. The default hop count is 30.
Time	Returns the current time on the target device.
SwitchControlProcessor	Switches from the current route processor to the specified route processor. If the target does not exist, the RPC returns an error message.
CancelReboot	Cancels any pending reboot request.
KillProcess	Stops an OS process and optionally restarts it.

File RPCs

The RPCs are used to perform key operations at the file level such as reading the contents if a file and its metadata. The **file.proto** file is available in the Github repository.

RPC	Description
Get	Reads and streams the contents of a file from the target device. The RPC streams the file as sequential messages with 64 KB of data.
Remove	Removes the specified file from the target device. The RPC returns an error if the file does not exist or permission is denied to remove the file.
Stat	Returns metadata about a file on the target device.
Put	Streams data into a file on the target device.

RPC	Description
TransferToRemote	Transfers the contents of a file from the target device to a specified remote location. The response contains the hash of the transferred data. The RPC returns an error if the file does not exist, the file transfer fails or an error when reading the file. This is a blocking call until the file transfer is complete.

Certificate Management (Cert) RPCs

The RPCs are used to perform operations on the certificate in the target device. The **cert.proto** file is available in the Github repository.

RPC	Description
Rotate	Replaces an existing certificate on the target device by creating a new CSR request and placing the new certificate on the target device. If the process fails, the target rolls back to the original certificate.
Install	Installs a new certificate on the target by creating a new CSR request and placing the new certificate on the target based on the CSR.
GetCertificates	Gets the certificates on the target.
RevokeCertificates	Revokes specific certificates.
CanGenerateCSR	Asks a target if the certificate can be generated.
LoadCertificateAuthorityBundle	Loads a bundle of CA certificates on the target. This CA certificate bundle is used to verify the client certificate when mutual TLS is enabled.

Interface RPCs

The RPCs are used to perform operations on the interfaces. The **interface.proto** file is available in the Github repository.

RPC	Description
SetLoopbackMode	Sets the loopback mode on an interface.
GetLoopbackMode	Gets the loopback mode on an interface.
ClearInterfaceCounters	Resets the counters for the specified interface.

Layer2 RPCs

The RPCs are used to perform operations on the Link Layer Discovery Protocol (LLDP) layer 2 neighbor discovery protocol. The **layer2.proto** file is available in the Github repository.

Feature Name	Description
ClearLLDPInterface	Clears all the LLDP adjacencies on the specified interface.

BGP RPCs

The RPCs are used to perform operations on the Link Layer Discovery Protocol (LLDP) layer 2 neighbor discovery protocol. The **bgp.proto** file is available in the Github repository.

Feature Name	Description
ClearBGPNeighbor	Clears a BGP session.

Diagnostic (Diag) RPCs

The RPCs are used to perform diagnostic operations on the target device. You assign each bit error rate test (BERT) operation a unique ID and use this ID to manage the BERT operations. The **diag.proto** file is available in the Github repository.

Feature Name	Description
StartBERT	Starts BERT on a pair of connected ports between devices in the network.
StopBERT	Stops an already in-progress BERT on a set of ports.
GetBERTResult	Gets the BERT results during the BERT or after the operation is complete.

MPLS RPCs

The RPCs are used to perform MPLS operations on the target device. The **mpls.proto** file is available in the Github repository.

Feature Name	Description
MPLSPing	Checks basic connectivity using MPLS ping operation. See RFC 4379. In Cisco IOS XR Release 7.5.4, the RPC supports `ldp_fec` and `rsvpte_lsp_name` destination types. The destination types `fec129_pwe` and `rsvpte_lsp` are not supported.
ClearLSP	Clears a single tunnel.
ClearLSPCounters	Clears the MPLS counters for the specified Label Switched Path (LSP).

Operating System (OS) RPCs

The OS service provides an interface for the OS installation on a target device. The RPCs replace the router software to upgrade the system. No concurrent installation is allowed on the same target. The **os.proto** file is available in the Github repository.

Feature Name	Description
Install	Transfers an OS package onto the target. **Note** Only Golden ISO installation is supported; RPM installation is not supported.
Activate	Sets the requested OS version as the version that is used at the next reboot. If booting up the requested OS version fails, the system recovers by rolling back to the previously running OS package.
Verify	Verifies the running OS version.

gNOI RPCs

The following examples show the representation of few gNOI RPCs:

Get RPC

Streams the contents of a file from the target.

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

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

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

Remove RPC

Remove the specified file from the target.

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

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

Reboot RPC

Reloads a requested target.

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

```
RPC start time: 21:12:49.811561
method: COLD
message: "Test Reboot"
subcomponents {
origin: "openconfig-platform"
elem {
name: "components"
}
elem {
name: "component"
key {
key: "name"
value: "0/RP0"
}
}
elem {
name: "state"
}
elem {
name: "location"
}
}
---------------------Reboot Request---------------------
RPC end time: 21:12:50.023604
```

Set Package RPC

Places software package on the target.

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

Reboot Status RPC

Returns the status of reboot for the target.

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

```
}
elem
name: "location"
}
}

RPC end time: 22:27:34.319618

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

CancelReboot RPC

Cancels any outstanding reboot

```
Request :
CancelRebootRequest
subcomponents {
origin: "openconfig-platform"
elem {
name: "components"
}
elem {
name: "component"
key {
key: "name"
value: "0/RP0/CPU0"
}
}
elem {
name: "state"
}
elem {
name: "location"
}
}

CancelRebootResponse

(rhel7-22.24.10) -bash-4.2$
```

KillProcess RPC

Kills the executing process. Either a PID or process name must be specified, and a termination signal must be specified.

```
KillProcessRequest
pid: 3451
signal: SIGNAL_TERM

KillProcessResponse
-bash-4.2$
```

gNOI Packet Link Qualification

Table 18: Feature History Table

Feature Name	Release Information	Feature Description
gNOI Packet Link Qualification	Release 24.2.11	You can now check and assess the reliability of the link speed and packet drops between the two network devices (generator and the reflector) by performing the gNOI packet-based link qualification service. This can be achieved by sending the packets from the generator to the reflector, and receiving the looped back packets from the reflector within a certain tolerance limit. The link transimmision rate and the link's capacity range for that interface can be obtained from the following gNSI Packet Link Qualification RPC messages: • `Capabilities`—Minimum and maximum rate of the transmission link • `Get`—Expected rate and actual rate of link transmission

The gRPC Network Operations Interface (gNOI) Packet Link Qualification service provides a way to certify link quality between a generator and a reflector device. The generator device generates test traffic and sends it out of the requested interface, maintaining counters of the sent, received, errored, and dropped packets. The reflector device loops back the traffic on the requested interface. The Packet-Based Link Qualification service verifies that the packets are sent and received on the requested interface. You can obtain the transmission rate and the link's capacity range for that interface from the gNSI Packet Link Qualification RPC messages: `Capabilities` and `Get`.

To view the packet link qualification specification, see the Github repository.

Table 19: Packet Link Qualification (PLQ) RPCs

RPC	Description
Capabilities	Fetches the capabilities of the device as a link qualification service. The capabilities result includes: • The roles supported on the device (Packet generator, Physical Medium Dependent (PMD) loopback reflector) • Information on whether the NTP synchronization is supported or not. • Information on whether the current device time is synchronized through NTP or not. • The Maximum number of results stored per interface

RPC	Description
Create	Creates a set of link qualifications on the device. Each element in a `Create` message specifies the following parameters: • A unique qualification ID • The interface on which to run the qualification • The endpoint type (the role of the device) • Role-specific configuration • Timing information in the form of either NTP-based or RPC-based timing For more information, see `Link Qualifications Based on Timing` table. **Note** Packet generator and PMD loopback roles are supported The packet injector and ASIC loopback roles are not supported.
Delete	Deletes a set of qualifications by their IDs. Stops all the running qualification tests listed and deletes their records from the device. The qualifications are automatically deleted from the device 24 hours either after successful completion or in the event of any error.
Get	Gets the status of each of the unique qualification IDs that you specify. For generator qualifications, it returns the number of packets sent, received, errored, dropped, and the expected and achieved rate in bytes per second. This data isn't present for reflector qualifications.
List	This RPC lists all the qualifications on the device.

Link Qualifications Based on Timing

When you run the `Create RPC` (see table `Packet Link Qualification (PLQ) RPCs`), it creates a set of link qualifications based on either it's NTP-based or RPC-based timing.

For both NTP-based and RPC-based timings, the qualification start time must be set no earlier than the minimum setup duration from the current time, as specified in the `Capabilities` RPC (see table `Packet Link Qualification (PLQ) RPCs`) response message.

NTP-based timing specifies:

- Specific start time
- Specific end time
- Teardown time

RPC-based timing specifies:

- Presync duration (duration from the current time to when the setup should start)

- Setup duration
- Qualification duration
- Postsync duration (duration from the end of the qualification to when the teardown should start)
- Teardown duration

gRPC Network Security Interface

Table 20: 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**: 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 88. 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 21: 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
```

gNSI Acctz Logging

Table 22: Feature History Table

Feature Name	Release Information	Feature Description
gNSI Acctz Logging	Release 24.3.1	Introduced in this release on: Fixed Systems(8200, 8700); Centralized Systems (8600); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) You can now log and monitor AAA (Authentication, Authorization, and Accounting) accounting of gRPC operations and CLI accounting data through gNSI Acctz for effective management of network for better performance and resource utilization. You can also configure the number of gNSI accounting records that can be streamed. Previously, you could monitor the AAA accounting data through syslog only. The feature introduces these changes: **CLI:** • **grpc aaa accounting queue-size** • **show gnsi acctz statistics** To view the specification of gNSI Accounting (Acctz) RPCs and messages, see the Github repository.

gNSI Acctz Data Logging

The gNSI accounting (Acctz) is a gNSI accounting protocol that collects and transfers accounting records from a router to a remote collection service over a gRPC transport connection.

Starting from Release 24.3.1, you can log gRPC AAA accounting data through gNSI accounting (Acctz). The gNSI Acctz data is logged, stored in accounting records, and send to gNSI client for monitoring purposes. These gNSI Acctz accounting records contain

- users' login or logout times,
- network access resources such as interface IP and port, and
- duration of each session.

The gNSI Acctz logging can be done using the RecordSubscribe() gRPC request to a router. For more information on the RecordSubscribe() RPC, see the GitHub repository.

gNSI Acctz Logging Stream Capacity

The gNSI Acctz logs are recorded in a queue, maintaining a history of the 10 most recent records. When the accounting queue is full and no gNSI Acctz collectors are connected, the stream drops the records. Besides the 10 records stored for streaming, up to 512 additional records are stored during processing. As new records arrive, the data stream continues until the gNSI session ends or an error occurs, such as a client disconnection due to network issues or the server going down. If the server's output buffer remains full for an extended period, new records are dropped until the collector starts receiving them.

When the queue reaches its full capacity, the system automatically replaces the oldest records with the newest ones. The router then transmits this logged information through gNSI to gNSI client for real-time monitoring purposes. You can configure the queue size using the **grpc aaa accounting queue-size** command.

Supported Records for gNSI Acctz Logging

gNSI Acctz logging system supports Command and gRPC service records.

Table 23: CLI and gRPC Accounting Records

Command Services Accounting Records	gRPC Services Accounting Records
The command accounting records are generated for the commands executed in CLI mode and sent to gNSI Acctz collectors. The details logged include: • **Session Info:** remote/local IP addresses, remote/local ports, and channel ID. • **Authentication details**: Identity, privilege level, authentication status (PERMIT/DENY), and the cause of denial (if applicable). • **Command** and **Command status**: authentication status (PERMIT/DENY). • **Timestamp:** The time when the event was generated.	The gRPC accounting records are generated for the RPCs executed by gRPC services and sent to gNSI Acctz collectors. The details logged include: • **Session Info**: remote/local IP addresses, remote/local ports, and channel ID. • **Authentication details**: Identity and privilege level. • **RPC Service Request**: Service type, RPC name, payload, and configuration metadata. • **gRPC Service Status**: PERMIT/DENY. • **Timestamp:** The time at which the event was generated.

Default Behavior and Verification of gNSI Acctz Logging

By default, gNSI Acctz records are logged when the configuration is enabled. You can verify the gNSI Acctz using **show gnsi state**, **show gnsi acctz statistics**, and **show aaa accounting statistics** commands.

Configure gNSI Acctz Logging

Monitor AAA information through gNSI Acctz logs.

Step 1 Monitor gNSI state in the router.

Example:

```
Router# show gnsi state
Wed Jun 26 09:26:39.035 UTC
----------------GNSI state--------------
Global:
    Main Thread cerrno              : Success
    Acctz Thread cerrno             : Success
    State                           : Active
    RDSFS State                     : Active
```

Step 2 Obtain gRPC port number.

Example:

```
show grpc
Tue Aug 13 14:21:50.995 IST

Server name                             : DEFAULT
Address family                          : dual
Port                                    : 57400

Service ports
  gNMI                                  : none
  P4RT                                  : none
  gRIBI                                 : none

DSCP                                    : Default
TTL                                     : 64
VRF                                     :
Server                                  : enabled
TLS                                     : disabled
TLS mutual                              : disabled
Trustpoint                              : none
Certificate Authentication              : disabled
Certificate common name                 : ems.cisco.com
TLS v1.0                                : disabled
Maximum requests                        : 128
Maximum requests per user               : 10
Maximum streams                         : 32
Maximum streams per user                : 32
Maximum concurrent streams              : 32
Memory limit (MB)                       : 1024
Keepalive time                          : 30
Keepalive timeout                       : 20
Keepalive enforcement minimum time      : 300

TLS cipher suites
  Default                               : none
  Default TLS1.3                        : aes_128_gcm_sha256
                                        : aes_256_gcm_sha384
                                        : chacha20_poly1305_sha256

  Enable                                : none
  Disable                               : none

  Operational enable                    : none
  Operational disable                   : none
Listen addresses                        : ANY
```

Step 3 Configure gNSI queue size.

Example:

```
Router# configure
Router(config)# grpc aaa accounting queue-size 30
Router(config)# end
```

Step 4 Monitor gNSI Acctz statistics in the router.

Example:

```
Router# show gnsi acctz statistics
Tue Aug 13 05:57:24.210 UTC
SentToAAA Queue:
  Grpc services:
    GNMI:    4998 sent, 0 dropped
    GNOI:    0 sent, 0 dropped
    GNSI:    2 sent, 0 dropped
    GRIBI:   0 sent, 0 dropped
```

```
    P4RT:    0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 5000
    Total Drops: 0

Streams:
  Grpc services:
    GNMI:    4996 sent, 2 dropped
    GNOI:    0 sent, 0 dropped
    GNSI:    1 sent, 0 dropped
    GRIBI:   0 sent, 0 dropped
    P4RT:    0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 4997
    Total Drops: 2
  Cmd services:
    CLI:     3 sent, 0 dropped
  Stats:
    Total Sent: 3
    Total Drops: 0
Router#
```

Step 5 Provide port and IP address to the Acctz gNSI client.

Example:

```
acctz_collector -server_addr 192.0.2.111:57400 -username <user name> -password <passwod> -dieafter
600

--------------- gSNI Remote Collector ---------------
2024/08/25 22:59:13    Connecting to gNSI Server.
2024/08/25 22:59:13    gNSI Server connected.
2024/08/25 22:59:13    Started new acctz client.
2024/08/25 22:59:13    Initiate Acctz RecordSubscribe with server .
2024/08/25 22:59:13    Stream started
2024/08/25 22:59:13 Waiting for response from server.
```

Step 6 Verify the accounting record from the router.

Example:

gNSI Acctz RPC RecordSubscribe() response to the Acctz gRPC client

```
session_info:
{
  local_address:"192.0.2.111"
  local_port:57400
  remote_address:"192.0.2.1"
  remote_port:44374
  ip_proto:6
  user:
    {
      identity:"lab"
    }
}
timestamp:
{
  seconds:1718971022 nanos:105825300
}
grpc_service:
{
  service_type:GRPC_SERVICE_TYPE_GNSI
```

```
  rpc_name:"/gnsi.acctz.v1.AcctzStream/RecordSubscribe" payload_istruncated:true
  authz:
  {
    status:AUTHZ_STATUS_PERMIT
  }
}
```

AAA Accounting Statistics

```
Router# show aaa accounting statistics
Sat Aug 17 17:10:43.055 UTC
Successfully logged events:
Total events: 0
XR CLI: 0
XR SHELL: 0
GRPC:
GNMI: 0
GNSI: 2
GNOI: 0
GRIBI: 0
P4RT: 0
SLAPI: 0
NETCONF: 0
SysAdmin:
CLI: 0
SHELL: 0
Host:
SHELL: 0

Errors:
Invalid requests: 0

Max. records in buffer: 100
Total records in buffer: 0
Router#
```

gNSI Credentialz Update

Table 24: Feature History Table

Feature Name	Release Information	Description
gNSI Credentialz Update	Release 24.2.11	To improve communication confidentiality and security, you can now update or rotate account-specific and host-specific SSH credentials on a router. You can access the latest SSH credentials through the gNMI credentialz RPC. The updated SSH credentials encompass passwords, host keys, and certificates. To view the specification of gNSI credentialz RPCs and messages, see the Github repository.

Rotation is the process of changing or updating SSH credentials such as passwords, keys, or certificates in a network. You can now update the account-related and host-related SSH credentials through the gNSI credentialz RPC when the router is up and running.

gNSI Rotate Credentialz RPC

Starting from Release 24.2.1, Cisco IOS XR supports four RPCs to change the existing SSH credentials.

gNSI Rotate Credentialz RPC	Run This When	For More Information
RotateAccountCredentials	You want to specify an SSH authentication service policy for the network element. If the policy is valid, it replaces the existing policy.	See, Rotate Account Credentials
RotateHostParameters	You want to change both the Certificate Authority (CA) public key and the key and certificate used by the SSH server.	See, Rotate Host Parameters
CanGenerateKey	You want to check whether the target can generate a public or private key pair.	See, CanGenerateKey
GetPublicKeys	You want to get the current public keys from the host. It returns each configured key in the provided list.	See, GetPublicKey

Rotate Account Credentials

This RPC automates secure credential rotation on routers, updating passwords and SSH keys to enforce security and prevent unauthorized access. It updates the user-specific authorized keys, authorized principles, invalidates old credentials, logs activities, and notifies stakeholders, enhancing overall network security.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The following table outlines the messages that `Rotate Account Credentials` RPC supports, along with their descriptions.

Message	Description
AuthorizedKeysRequest	This message defines the authorized key list for password-less SSH accepted by the router's SSH service. The gNSI client dispatches an `AuthorizedKeysRequest` to the router to update or replace credentials on the SSH service. The router responds with a `AuthorizedKeysResponse` message to the gNSI client. It supports the following keys: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519
AuthorizedUsersRequest	This message performs a user authorization check. User authorization can be done using both static and dynamic methods. Static Authorization: You can perform static authorization based on a principal name (unique identifier for a user) using Cisco SSH. For static authorization, use the `AuthorizedUsersRequest` message. Dynamic authorization: For dynamic authorization, use the `AuthorizedPrincipalCheckRequest` message. For details, see Rotate Host Parameters, on page 99 CiscoSSH supports the user authorization using AuthorizedPrincipalsFile. AuthorizedPrincipalsFile contains pairs of account names and their corresponding principal names that the router recognizes for certificate-based authentication. For more details, see AuthorizedPrincipalsFile

Rotate Host Parameters

The `RotateHostParameters` RPC updates and verifies host account credentials on network devices to enhance security and ensure stable SSH access. If updates fail, the system either adopts new credentials after successful validation or reverts to the old ones to maintain uninterrupted access. The router automatically falls back to prevent lockouts and preserve network integrity.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The following table outlines the messages that `Rotate Host Parameters` RPC supports, along with their descriptions.

Message	Description
CA public key	The SSH server uses the `CA public key` message to verify the gNSI client certificates presented during connection establishment. Without Host Identity Based Authorization (HIBA), the following keys are supported: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519
Server keys	The `Server keys` message includes host keys and router certificates that serve as credentialz for the gNSI client. If the host keys are generated externally, they must be specified in the `Server keys` request. It supports the following keys: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519 It supports the following router certificates: • Router certificates with HIBA Support • ssh-rsa-cert-v01@openssh.com • Router certificates without HIBA support: • ecdsa-sha2-nistp256-cert-v01@openssh.com • ecdsa-sha2-nistp521-cert-v01@openssh.com • ssh-ed25519-cert-v01@openssh.com • rsa-sha2-256-cert-v01@openssh.com • rsa-sha2-512-cert-v01@openssh.com
Generate key	The `Generate Key` message is used for host key management in SSH. When the host keys are generated by the router, this message triggers the creation of new host keys for SSH host key management. The `Generate key` message supports the following keys: It supports the following keys: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519

Message	Description

AllowedAuthenticationRequest	The `AllowedAuthenticationRequest` message specifies the permissible authentication methods for the gNSI client authentication.
	The supported authentication methods are as follows:
	• Keyboard interactive
	• Password-based
	• Pubkey-based
	  • OpenSSH certificate-based
	  • Public key-based
	By default, the SSH server allows all authentication methods.

AuthorizedPrincipalCheckRequest	The `AuthorizedPrincipalCheckRequest` message supports the dynamic authorization of the user against the principal name using the OpenSSH or CiscoSSH. Setting the TOOL_HIBA_DEFAULT flag prompts the router to use the HIBA binary for dynamic authorization. Un setting the HIBA_DEFAULT flag switches the router to use a static authorization. Dynamic Authorization: You can enforce the user for authorization check using HIBA. **Note** The support is only for ssh-rsa-cert-v01@openssh.com CiscoSSH supports AuthorizedPrincipalCheck using `AuthorizedPrincipalsCommand` and `AuthorizedPrincipalsCommandUser` AuthorizedPrincipalsCommand: This command generates the list of allowed certificate principals by executing a HIBA binary (By setting the TOOL_HIBA_DEFAULT flag). AuthorizedPrincipalsCommandUser: This command specifies the user account under which the system executes the `AuthorizedPrincipalsCommand`. For more details on the specification, see AuthorizedPrincipalsCommandUser

CanGenerateKey

This RPC checks if the router can generate a public or private key pair.

It supports the following key pairs:

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

GetPublicKey

This RPC gets the available public keys from the router and displays them. It supports the following keys:

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

Manage certificates using Certz.proto

Table 25: Feature History Table

Feature Name	Release Information	Feature Description
Manage certificates using Certz.proto	Release 24.1.1	Instead of using multiple RPCs, Certz.proto provides a bidirectional Rotate RPC to replace, revoke, or load a certificate. It also provides additional APIs to install Public Key Infrastructure (PKI) entities such as like identity certificates, trust-bundles, and Certificate Revocation Lists (CRLs) for a gRPC Server. This feature introduces the following changes: CLI: • **grpc gnsi service certz ssl-profile-id** • **show grpc certificate** Yang Data Models: • Cisco-IOS-XR-man-ems-cfg.yang (see Github, YANG Data Models Navigator)

Certz RPCs

The Certz RPCs are specific methods used for executing operations on the certificate that resides in the target device. The **certz.proto** file is available in the Github repository.

In cert.proto, a certificate identifier differentiates between leaf certificates. However, the CA bundle lacks an identifier, meaning a new request to load a bundle could overwrite the existing one. On the other hand, in certz.proto, entities like Certificate, CA bundle, key, CRL, and authentication policy are tied to a unique SSL profile.

In cert.proto, a certificate identifier differentiates between leaf certificates. However, the CA bundle lacks an identifier, meaning a new request to load a bundle could overwrite the existing one. On the other hand, in certz.proto, entities like Certificate, CA bundle, key, CRL, and authentication policy are tied to a unique SSL profile.

The certz.proto differs from the cert.proto in the way that it handles the upload of all entities. While in cert.proto, separate RPCs are used to replace, load, and revoke a certificate, in certz.proto, a single Rotate() RPC is used to upload all entities at once. This includes the certificate, the key, the CA bundle, and the CRL.

In addition to these features, certz.proto also provides support for different cryptographic algorithms, including Rivest-Shamir-Adleman (RSA), Elliptic Curve Digital Signature Algorithm (ECDSA), and ED25519, a public-key signature system.

These functionalities make certz.proto a comprehensive solution for managing SSL profiles, providing a streamlined process for handling cryptographic entities and algorithms.

Note If neither cert.proto nor certz.proto is configured, then tls trustpoint data is considered for certificate management.

The following table describes the RPCs supported under Certz.proto.

Table 26: Certz RPCs

RPC	Description
AddProfile	AddProfile is part of SSL profile management. It allows adding a new SSL profile. When an SSL profile is added, all its elements, that is, certificate, CA trusted bundle and a set of certificate revocation lists are NULL/Empty. So, before an SSL profile can be used these entities have to be 'rotated' using the `Rotate()` RPC. **Note** An attempt to add an already existing profile is rejected with an error.
Rotate	Rotate replaces/adds an existing device certificate and/or CA certificates (trust bundle) or/and a certificate revocation list bundle on the target. The new device certificate can be created from a target-generated or client-generated CSR (Certificate Signing Request). In the latter case, the client must provide the corresponding private key with the signed certificate.
DeleteProfile	DeleteProfile is part of SSL profile management. It allows for removing an existing SSL profile. **Note** An attempt to delete a not existing profile results in an error. The profile used by the gRPC server can’t be deleted and an attempt to remove it will be rejected with an error.
GetProfileList	GetProfileList is part of SSL profile management. It allows for retrieving a list of IDs of SSL profiles present on the target.
CanGenerateCSR	An RPC to ask a target if it can generate a CSR.

SSL Profile

An SSL profile is a named set of SSL settings that determine how end-user systems connect to or from SSL-based applications or interfaces. The settings in an SSL profile include information about the version of SSL/TLS to be used, certificates, keys, and other parameters related to SSL/TLS communication. By using profiles, administrators can manage and apply these settings more easily across multiple applications or connections.

Here are some key-points regarding SSL profile:

- SSL profiles logically groups certificate, private key, Certificate Authority chain of certificates (a.k.a. a CA trust bundle) and a list of Certificate Revocation Lists into a single set that then can be assigned to a gRPC server.
- There's at least one profile present on a target - the one that is used by the gRPC server. Its ID is gNxI but when the ssl_profile_id field in the RotateCertificateRequest message isn't set (or set to an empty string) it also refers to this SSL profile by default.
- You can't remove the gRPC SSL profile (gNxI).

Configure gNSI Certz

Before you begin

- Ensure you've created and stored SSL-Profile at `cd/misc/config/grpc/gnsi/certz/ssl_profiles/`

Step 1 Create SSL-Profile using AddProfile RPC.

Step 2 Rotate SSL-profile using Rotate RPC. You can't rotate SSL-profile using a command line interface.

Step 3 Activate the profile using **grpc gnsi service certz ssl-profile-id.**

Example:

Router (config-grpc) #**gnsi service certz profile ssl-profile id <ssl-profile-name>**

Step 4 Verify that certz.proto is configured using the **show grpc certificate**.

Example:

```
Router#show grpc certificate
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 32 (0x20)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost,O=OpenConfig,C=US
        Validity
            Not Before: Nov  8 08:49:38 2023 GMT
            Not After : Mar 22 08:49:38 2025 GMT
        Subject: CN=ems,O=OpenConfig,C=US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:ea:6a:6c:25:be:9f:15:71:ce:74:89:03:ec:ef:
                    0b:3b:de:58:a8:7e:28:b8:cf:b3:82:91:b4:5c:42:
                    e7:d8:28:98:35:bd:35:60:a7:4e:f8:77:02:46:5f:
                    27:a4:16:cf:3c:e3:24:28:69:9c:22:1e:e3:52:96:
                    71:87:7c:40:0c:1f:dd:30:ea:dc:40:ca:93:00:54:
                    5e:de:20:54:5b:f4:2f:9f:19:6f:71:61:28:69:3d:
```

```
                    97:26:ab:e1:5f:53:3c:f1:a2:c3:14:f4:01:90:1a:
                    e3:08:7b:51:c9:5d:aa:6d:eb:99:a4:08:97:d3:72:
                    8c:86:a3:f3:b3:77:10:72:e7:a9:3b:fc:38:65:3d:
                    41:1a:f5:cf:3e:a0:d8:17:d6:d5:53:86:49:a3:dc:
                    cc:3a:d9:6d:46:25:b0:f9:3b:98:fa:2f:98:09:08:
                    51:ac:2c:b1:43:c4:b7:96:3e:4e:4e:a6:a5:36:1f:
                    1f:0f:6a:6a:1a:ea:72:6e:74:90:21:05:fb:26:df:
                    81:0d:96:e7:13:94:62:2b:ce:3c:7c:de:32:f4:d9:
                    fa:24:ce:f5:b2:0f:d3:f7:4b:6b:ee:bd:cf:ac:a6:
                    ed:69:37:fc:d3:4f:3b:46:8b:1b:62:4d:3b:60:30:
                    74:68:50:4e:48:35:5f:15:66:9a:01:7c:37:1f:e1:
                    5a:8a:d9:c0:2c:3e:12:fd:71:30:13:b8:b7:16:98:
                    03:27:6d:45:c4:0f:34:fd:f1:aa:29:8e:c1:63:ac:
                    57:04:f6:a7:83:83:06:45:dc:0f:f9:de:f9:1e:b6:
                    d8:5a:bc:3a:98:f8:ac:b0:be:3f:87:df:8c:5e:47:
                    12:ca:77:70:26:14:02:14:79:fa:6f:1f:ab:ee:06:
                    2c:83:93:e4:22:db:37:83:90:c1:72:5b:36:78:1b:
                    6d:0a:06:72:76:dc:89:df:86:89:43:54:03:55:bd:
                    fc:a0:9a:d6:8e:5d:22:87:a2:32:19:35:c8:17:4e:
                    1c:1b:5e:81:9d:a5:67:9e:a7:ed:06:e8:e2:91:f1:
                    ae:f9:19:b1:ae:a8:e6:66:14:2c:6d:a6:c3:0f:8b:
                    7f:ef:c0:60:cb:c2:52:a5:46:1e:a4:20:52:f8:93:
                    93:2b:02:23:98:90:81:b3:e6:c4:4e:8f:85:a6:ff:
                    4e:8e:dd:6c:12:ea:db:58:7f:3c:66:c4:38:96:44:
                    d1:5b:da:c2:66:6a:4e:97:4d:99:59:9f:24:a0:4a:
                    57:b6:9d:69:22:f7:5a:10:cb:96:bc:58:ca:96:0e:
                    ab:b0:4d:14:da:03:e1:d3:24:c1:f2:bd:40:32:20:
                    82:66:4d:78:4b:13:c6:bd:66:a9:83:2f:15:29:7e:
                    11:95:37
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, TLS Web Server Authentication
            X509v3 Authority Key Identifier:
                keyid:0A:A8:9A:6A:23:34:AE:CA:96:00:2C:F3:04:38:14:E3:D4:8D:77:BD

            X509v3 Subject Alternative Name:
                DNS, IP Address:64.103.223.56
    Signature Algorithm: sha256WithRSAEncryption
         b9:89:ec:60:3d:8d:7d:9c:dc:08:56:89:99:44:92:98:45:b6:
         97:ba:e3:e5:f2:48:b2:44:8d:db:23:bb:a1:c0:62:79:78:18:
         d7:55:f6:4a:67:5b:75:e0:c0:0b:52:51:07:36:d5:6c:c7:67:
         48:86:8d:dd:70:1c:9f:7c:a1:7b:aa:a5:4e:e1:ad:cf:4c:e5:
         81:db:92:cf:88:70:5a:1c:8d:de:0d:e8:b3:05:de:b9:04:4d:
         23:e1:de:66:e5:08:bd:2e:31:0a:07:a6:c0:00:3a:38:2f:00:
         cd:cf:be:e2:1f:12:9f:8a:44:8d:2d:24:d5:d3:bb:9e:db:70:
         bf:89:ea:0c:31:b4:b2:fc:3d:73:f5:17:09:07:54:ab:2f:23:
         cb:66:0e:0e:7a:9e:21:bf:1e:bf:07:f1:fc:09:88:23:4e:2d:
         5d:08:35:16:cd:07:df:25:34:7f:42:0a:dc:6f:d0:ec:9d:99:
         72:d8:5f:d6:7e:6f:cc:67:4d:d7:b9:b8:c8:56:75:db:56:1e:
         03:1b:6d:37:21:4d:e0:f1:e2:80:99:40:24:24:f2:e4:9b:7e:
         6c:bc:f7:f9:3a:b6:fc:8e:dd:9a:cd:dd:88:15:d7:46:71:d2:
         11:20:86:8f:ea:c5:a8:e8:4e:b6:ef:9b:06:5b:b1:c4:11:36:
         38:7a:63:8e:1a:a6:a8:f8:bb:7d:0b:a6:f2:89:49:94:ac:0c:
         8b:c4:fc:02:e8:b2:b8:27:bc:70:95:32:83:09:f5:de:68:34:
         3f:a4:5a:73:dc:92:15:2c:0e:ab:46:dd:13:06:98:aa:08:2d:
         b8:37:a0:52:4b:ba:f7:be:ed:68:cd:fb:67:3b:66:ea:16:85:
         61:75:cf:06:85:a0:06:e8:4a:3e:63:72:c1:79:c7:fd:d4:85:
         74:d8:ea:66:d3:42:74:e2:fb:7c:9e:93:4b:24:2f:ad:c5:13:
         bc:eb:83:f7:6d:3e:53:9a:ec:16:85:b7:b5:6c:77:48:53:7e:
         19:2e:48:2d:83:35:7b:b9:66:5e:12:b4:f3:ee:e8:b2:3b:ba:
         18:46:91:b0:f9:6f:b0:d5:17:a8:de:5c:a0:0e:35:85:7b:c0:
```

```
e3:79:06:fa:ad:8e:f2:28:ab:09:19:b7:f0:f3:9e:cb:94:93:
b7:04:63:74:82:c3:71:3b:16:8b:58:c7:fa:ff:ff:2a:97:91:
e7:1d:06:ab:0a:6c:cc:a0:41:31:54:f2:e7:db:a3:b5:22:c4:
ab:ec:e2:5d:86:e6:ac:a5:c6:e2:0e:15:44:a2:32:42:3d:07:
65:0a:0d:58:2e:22:3c:7b:e3:e8:8e:2e:60:47:f0:60:04:89:
64:65:fc:fc:74:dd:4d:7f
```

P4Runtime

Table 27: Feature History Table

Feature Name	Release Information	Description
P4Runtime to Manage Traffic Operations	Release 7.10.1	With this release, the router supports Programming Protocol-Independent Packet Processors Runtime (P4), a gRPC-based service, to program the data plane elements for network operations such as sending and receiving packets between the router and the P4Runtime controller using packet I/O messages. This feature introduces the following commands: **CLI:** • **grpc p4rt** • **grpc p4rt interface** • **grpc p4rt location** • **show p4rt devices** • **show p4rt interfaces** • **show p4rt state** • **show p4rt stats** • **show p4rt trace** **YANG Data Model:** `openconfig-p4rt.yang` OpenConfig data model (see GitHub, YANG Data Models Navigator)

P4Runtime is a control plane specification to manage the data plane elements of a device. It defines the navigation and management of packets through data plane blocks using P4Runtime APIs. These blocks can be managed to perform the following set of traffic operations between the P4Runtime controller and the router:

- Send or receive packets using PacketOut and PacketIn I/O messages—`StreamMessageRequest`, `StreamMessageResponse` and `StreamError` messages.
- Elect the primary controller using the `MasterArbitrationUpdate` message.
- Read and write forwarding table entries, protocol headers, counters, and other P4 entities.

For more information about how controllers can connect to the router and program P4-defined functionalities, see P4RT specification.

Configure P4RT to Manage Packets

Configure P4RT to send or receive packets between one or more controllers and the router.

Step 1 Enable P4Runtime.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#p4rt
Router(config-grpc-p4rt)#commit
```

Step 2 Assign a unique P4 numeric identifier to the required physical port on the router. The controller uses this port ID as an alias to identify the interface through which the packets are sent or received with ingress or egress metadata.

Example:

```
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/24 port-id 3
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/25 port-id 6
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/26 port-id 7
```

The `port-id` is a unique 32-bit identifier. The range is 1 to 4294967039.

Step 3 Assign a unique P4 device identifier to each Network Processing Unit (NPU) in the system.

Example:

```
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 0 device-id 1000000
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 1 device-id 1000001
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 0 device-id 1000002
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 1 device-id 1000011
Router(config-grpc-p4rt)#commit
Router(config-grpc-p4rt)#end
```

The `device-id` is a unique 64-bit identifier. The range is 1 to 18446744073709551615. The `npu-id` represents a NPU identifier within a line card and the value ranges from 0 to 7.

The controller or the P4Runtime agent, which can be external or internal to the router, can use the `port-id` and `device-id` to inject packets and request to send certain packet types. For example, P4Runtime supports the ability to configure Access Control Lists (ACLs) in order to redirect packets with TTL value 1 to the controller. When the router receives a packet with that TTL value, the packet is sent to the controller with the details such as packet received from `device-id x`, `port-id y` and the packet is being sent to `port-id z`.

For more information about programming the router using P4Runtime, see P4RT specification.

IANA Port Numbers For gRPC Services

Table 28: Feature History Table

Feature Name	Release Information	Description
IANA Port Numbers For gRPC Services	Release 24.1.1	You can now efficiently manage and customize port assignments for gNMI, gRIBI, and P4RT services without port conflicts. This is possible because Cisco IOS XR now supports the Internet Assigned Numbers Authority (IANA)-assigned specific ports for P4RT (Port 9559), gRIBI (Port 9340), and gNMI (Port 9339). You can now use both IANA-assigned and user-specified ports for these gRPC services across any specified IPv4 or IPv6 addresses. As part of this support, a new submode for gNMI in gRPC is introduced. This feature introduces the following changes: **CLI:** • **port (gRPC)** • **gnmi**

IANA (Internet Assigned Numbers Authority) manages the allocation of port numbers for various protocols. These port numbers help in distinguishing different services on a network. Service names and port numbers are used to distinguish between different services that run over transport protocols such as TCP, UDP, DCCP, and SCTP. Port numbers are assigned in various ways, based on three ranges: System Ports (0-1023), User Ports (1024-49151), and the Dynamic and/or Private Ports (49152-65535).

Earlier, the gRPC server configuration on IOS-XR allowed a usable port range of 10000-57999, with a default listening port of 57400 and all services registered to the gRPC server utilized this port for connectivity. Service-based filtering of requests on any of the ports was unavailable. Hence, the request for a specific service sent on a port designated to another service (for example, gRIBI request on gNMI port) was accepted.

From Cisco IOS XR Release 24.1.1, a new submode for gNMI is introduced in the configuration model to allow for service-level port customization. The existing gRPC configuration model includes submodes for P4RT and gRIBI. This submode will enable you to configure specific ports for gNMI, gRIBI, and P4RT services independently. You can configure gNMI, gRIBI, and P4RT services using the gRPC submode command to set the default port for each service. The **port** command under service submode, allows you to modify the port as needed, while adhering to the defined port range.

Disabling the **port** command will cause the service to use the default or IANA port.

You can set custom ports for gNMI, gRIBI, and P4RT services within the defined range, including default IANA ports like 9339, 9340, and 9559 (respectively). The gRPC service will continue to maintain its default port within the specified range (57344-57999). Any changes made to the gRPC default port will not impact the service port configurations for gNMI, gRIBI, and P4RT. Requests which are sent on a port designated for a specific service (example, gRIBI request on gNMI port) will be accepted. This flexibility allows for seamless communication across different service ports and the general gRPC port.

Starting from Release 24.2.1, the allowed port range is 1024-65535.

Configure gRPC Service-Level Port

To configure a default listening port for the gRPC services such as gNMI, gRIBI, and P4RT, use the respective service command (**gnmi**, **gribi**, or **p4rt**) under the gRPC configuration mode.

To specify a port number for gRPC, gNMI, gRIBI, and P4RT services within the defined range, use the **port** command under respective submodes.

Note The IANA port ranges are:

- System ports (Reserved): 0—1023
- Registered ports: 1024—49151
- Dynamic or Private or Ephemeral ports: 49152—65535

XR Ephemeral port range: 15232–57343

If the configured port is in the range of IANA registered ports (1024-49151) or XR ephemeral ports (15232-57343), a syslog is generated with a NOTICE to warn the user for a possible application conflict.

Resetting the port reverts to the default service port, and disabling the service stops listening on that port.

Configure the port number for a service.

The following examples display the service-level port configurations.

- **For gRPCservice:**

  This configuration creates a gRPC listener with the default or IANA ratified port of 57400.

  The allowed range is 1024-65535.

```
Router#config
Router(config)#grpc
Router(config-grpc)# commit
```

  Verify the listening port created for gRPC service.

```
Router#show running-config grpc
grpc
!
```

  The **port** command under gRPC submode allows the port to be modified in the port range or IANA ratified port.

```
Router# config
Router(config)# grpc port 2000
Router(config)# commit
```

  Verify the port number.

```
Router#show running-config grpc
grpc
  port 2000
!
```

- **For gNMI service:**

  This configuration creates a gRPC listener with the default or IANA ratified gNMI port of 9339.

  The allowed range is 1024-65535.

```
Router(config-grpc)#gnmi
Router(config-grpc-gnmi)#commit
```

  Verify the listening port created for gNMI service.

```
Router#show running-config grpc
grpc
  gnmi
!
```

  The **port** command under gNMI submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc)#gnmi
Router(config-grpc-gnmi)#port 9339
Router(config-grpc-gnmi)#commit
```

  Verify the port number.

```
Router#show running-config grpc
grpc
  gnmi
    port 9339
!
```

- **For P4RT service:**

  This configuration creates a gRPC listener with the default or IANA ratified P4RT port of 9559.

  The allowed range is 1024-65535.

```
Router(config-grpc)#p4rt
Router(config-grpc-p4rt)#commit
```

  Verify the listening port created for P4RT service.

```
Router#show running-config grpc
grpc
  p4rt
!
```

  The **port** command under P4RT submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc)#p4rt
Router(config-grpc-p4rt)#port 9559
Router(config-grpc-p4rt)#commit
```

  Verify the port number.

```
Router#show running-config grpc
grpc
  p4rt
    port 9559
!
```

- **For gRIBI service:**

  This configuration creates a gRPC listener with the default or IANA ratified gRIBI port of 9340.

  The allowed range is 1024-65535.

```
Router(config-grpc)#gribi
Router(config-grpc-gribi)#commit
```

Verify the listening port created for gRIBI service.

```
Router#show running-config grpc
grpc
  gribi
!
```

The **port** command under gRIBI submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc)#gribi
Router(config-grpc-gribi)#port 9340
Router(config-grpc-gribi)#commit
```

Verify the port number.

```
Router#show running-config grpc
grpc
  gribi
    port 9340
!
```

Unconfiguring the port command in a service

and

Unconfiguring a service under gRPC

- Unconfiguring the **port** command results in using the default port for the respective service.

  Example:

  Unconfiguring the **port** command will result in a gNMI service using the default gNMI port.

  ```
  Router(config-grpc)#gnmi
  Router(config-grpc-gnmi)#no port
  Router(config-grpc-gnmi)#commit
  ```

  Verify the service port configuration.

  ```
  Router#show running-config grpc
  grpc
    gnmi
  !
  ```

- Unconfiguring a service removes the listener for the respective port and no requests will be accepted on that port.

  Example:

  Unconfiguring gNMI disables the requests on port 9339.

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

  Verify the port configuration.

  ```
  Router#show running-config grpc
  grpc
  !
  ```

Configure Interfaces Using Data Models in a gRPC Session

Table 29: Feature History Table

Feature Name	Release Information	Description
Set Limit on Concurrent Streams for gRPC Server	Release 24.1.1	You can prevent potential security attacks by disallowing any single gRPC server client on Cisco IOS XR from consuming excessive resources and monopolizing connection resources, both of which can be potential attack vectors. Such prevention is possible because you now have the option to configure the gRPC server to limit the number of concurrent streams per gRPC connection. The feature introduces the **grpc max-concurrent-streams** command. **YANG Data Models:** • `Cisco-IOS-XR-man-ems-oper.yang` • `Cisco-IOS-XR-man-ems-cfg.yang` (see GitHub, YANG Data Models Navigator)

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.

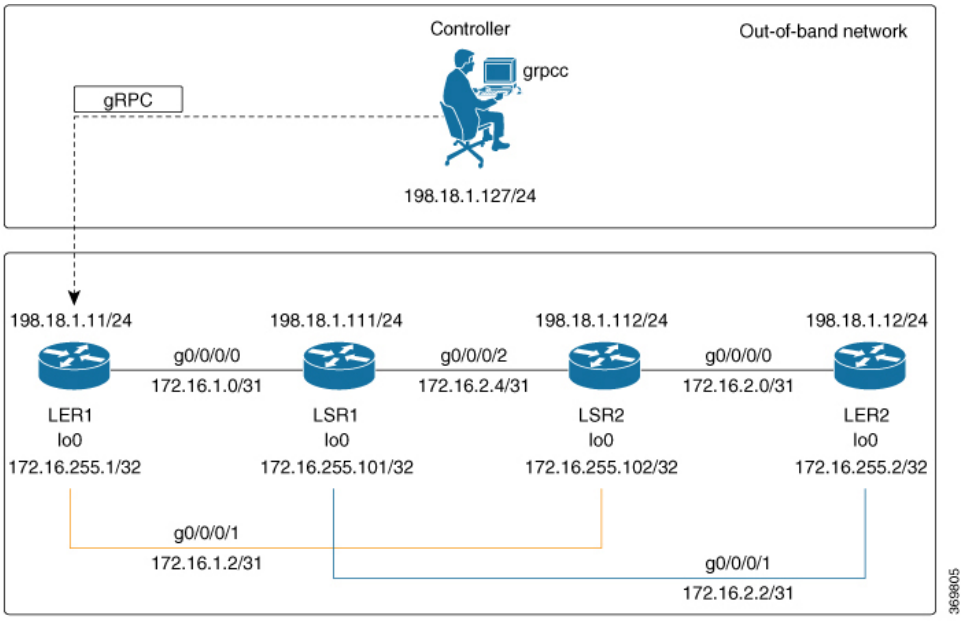
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network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

Starting Release 24.1.1, you can now configure IANA port numbers for specified gRPC services. To see the port numbers for the various gRPC services, see *Support IANA Port Numbers*.

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-concurrent-streams`: set the limit on the maximum concurrent streams per gRPC connection to be applied on the server.
- `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 48. 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 ''
```

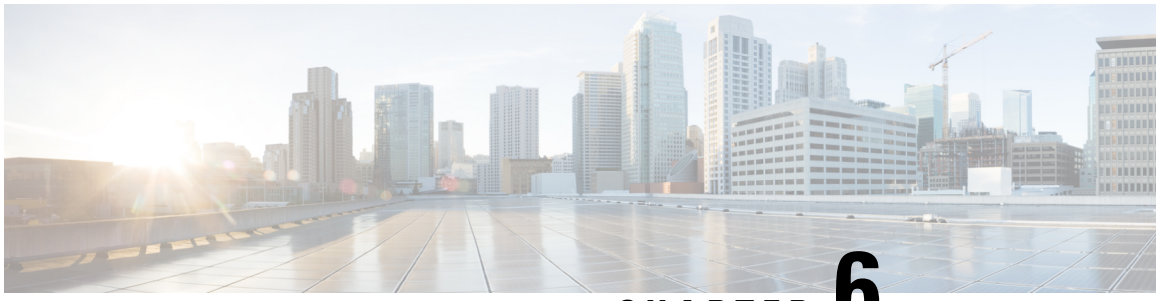
Step 3 Verify that the loopback interface and the ethernet interface on router LER1 are operational.

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 6

Use Service Layer API to Bring your Controller on Cisco IOS XR Router

Bring your protocol or controller on IOS XR router to interact with the network infrastructure layer components using Service Layer API.

For example, you can bring your controller to gain control over the Routing Information Base (RIB) tables and many more use cases.

Get to Know Service Layer API

Service Layer API is a model-driven API over Google-defined remote procedure call (gRPC).

gRPC enables you to bring your applications, routing protocols, controllers in a rich set of languages including C++, Python, GO, and many more.

Service Layer API is available out of the box and no extra packages required.

In IOS XR, routing protocols use RIB, the MPLS label manager, BFD, and other modules, to program the forwarding plane. You can expose these protocols through the service layer API.

Benefits

The Service Layer API gives direct access to the Network Infrastructure Layer (Service-Adaptation Layer). Therefore, you have the following advantages:

- **High Performance**: Direct access to the Network Infrastructure Layer, without going through a Network state database, results in higher performance than equivalent Management APIs.

  For example, Batch updates straight to the Label Switching Data Base (LSDB), the Routing Information Base (RIB) (over gRPC). The LSDB stores label-to-address mappings for efficient traffic routing in Label-switching routers. And, RIB contains the active and potential routes to various network destinations.

- **Flexibility**: The Service Layer API gives you the flexibility to bring your Protocol or Controller over gRPC.

- **Offload low-level tasks to IOS XR**: IOS XR infrastructure layer handles the following. Hence, you can focus on higher-layer protocols and controller logic:
  - Conflict resolution
  - Transactional notifications
  - Data plane abstraction

Components of Service Layer API

The following are the components of the Service Layer API architecture:

- **Functionality Verticals/Domains**: The verticals define the broader capability categories supported by the API. The following are the supported verticals. Each vertical supports data structure and RPCs defined in gpb
  - **Initialization**: Handles global initialization, sets up an event notification channel using GRPC streaming capabilities.

    The initialization RPCs are mandatory. Use the initialization RPCs to connect a client to the gRPC server on the router. Also, to send heartbeats and termination requests from the server to the client.
  - **IPv4, IPv6 Route (RIB)**: Handles route manipulations (add, update, delete) for a certain VRF.
  - **MPLS**: Handles allocation of label blocks and any incoming MPLS label mapping to a forwarding function.
  - **Interface**: Handles subscription of the registered clients to the interface state event notifications.
  - **IPv4, IPv6 BFD**: Manages BFD sessions, and corresponding BFD session state notifications.
- **Protobuf Schema/Model**: Use gRPC to model the service layer API.
- **gRPC**: gRPC utilizes GPB protobuf IDL by default to convert the models into bindings in various languages (c++, python, golang, and more). The gRPC server (running on the router) and the gRPC client use the generated bindings to serialize data and encode or decode the request or response between the server and the client.
- **Service Layer gRPC clients**: Based on the business needs, the gRPC clients for service layer can exist in one of the following ways:
  - On-box (agents runnning on their own sand-boxed third-party containers)
  - Off-box (within Controllers or other open-source tools)
- **gRPC Authentication Modes:**

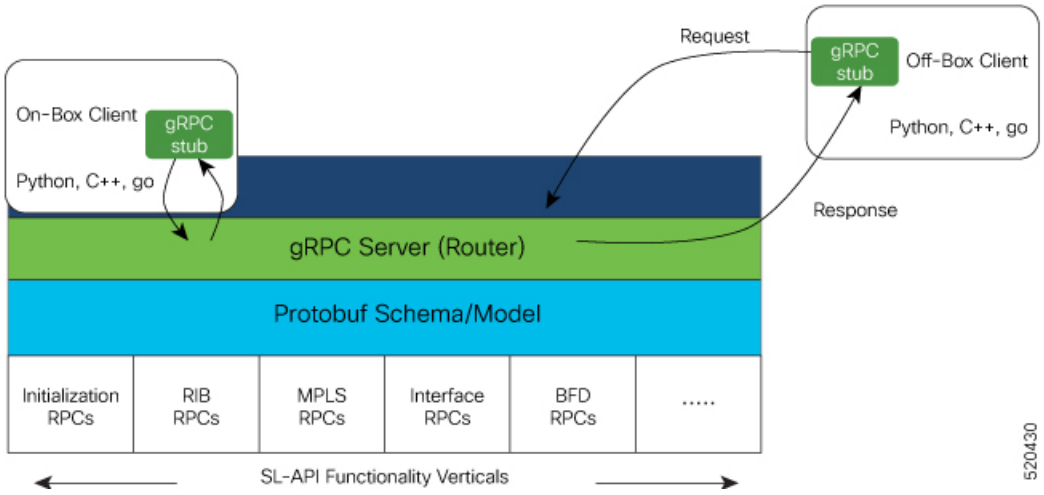
  gRPC supports the following authentication modes to secure communication between clients and servers. These authentication modes help ensure that only authorized entities can access the gRPC services, like gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device will authenticate the user and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

  The following table lists the authentication type and configuration requirements:

Table 30: Types of Authentication with Configuration

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

Figure 6: Components of Service Layer API



Bring your controller

To bring your controller on IOS XR, first, enable the service layer on the router and then write your Service Layer Client API.

1. Enable Service Layer, on page 122
2. Write Your Service Layer Client API

Enable Service Layer

Step 1 Enable the Service Layer.

Example:

```
Router#configure
Router(config)#grpc
Router(config-grpc)#port 57777
Router(config-grpc)#service-layer
Router(config-grpc)#no-tls
Router(config-grpc)#commit
```

The default port value for gNMI service port is 9339. You can set gNMI service port value from 57344 to 57999. Whereas, the default port value for gRIBI service port is default 9340. You can set gRIBI service port value from 57344 to 57999.

Step 2 Verify if the Service Layer is operational:

Example:

```
Router#show running-config grpc
Mon Nov  4 04:19:14.044 UTC
grpc
 port 57777
 no-tls
 service-layer
 !
!
```

Step 3 Verify the gRPC state.

Example:

```
Router#show service-layer state
Mon Feb 24 04:18:40.055 UTC
------------service layer state----------
config on:                    YES
standby connected :           NO
idt done:                     NO
blocked on ndt:               NO
connected to RIB for IPv4:    YES
connected to RIB for IPv6:    YES
Initialization state:         estab sync
pending requests:             0
BFD Connection:               UP
MPLS Connection:              UP
Interface Connection:         UP
Objects accepted:             NO
interface registered:         NO
bfd registered for IPv4:      NO
bfd registered for IPv6:      NO
```

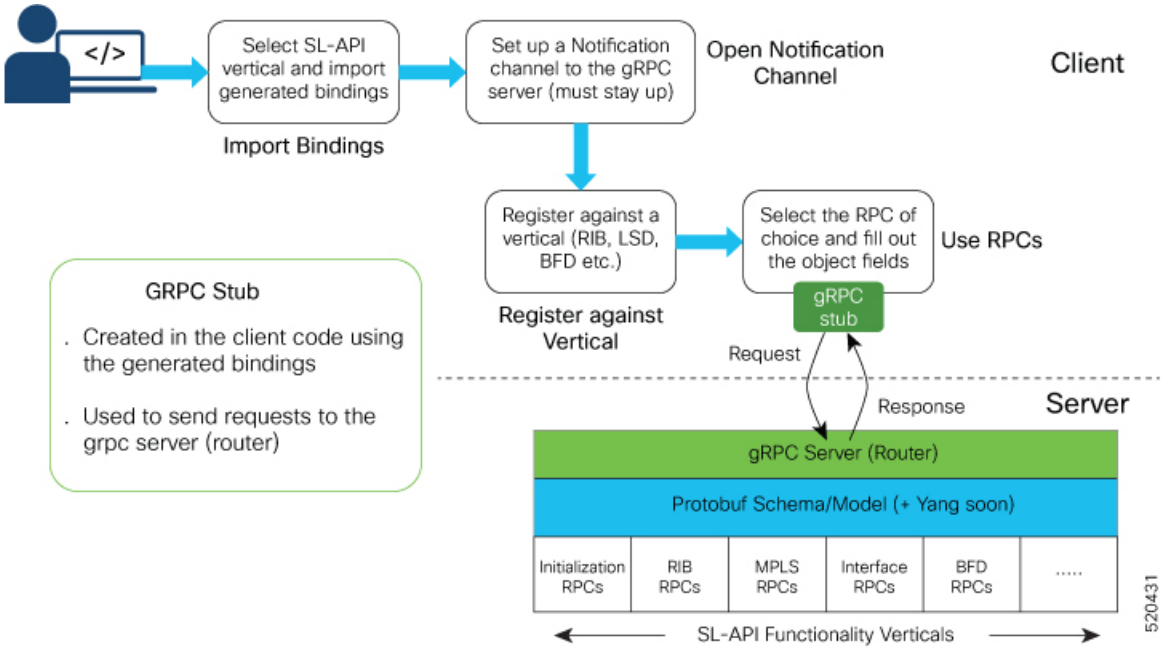
Write Your Service Layer Client API

You can write a Service Layer API based on your business needs. Follow these steps to write a Service Layer API client for a particular functionality vertical.

- **Import Bindings**: After generating the bindings, import the binding in your code.
- **Open Notification Channel:** Utilize the initialization functionality vertical to create a notification channel to register the client to the gRPC server running on the router.
- **Register against Vertical**: Register for a functionality vertical to utilize an RPC using the registration RPC before making calls. The system rejects any calls without prior registration.
- **Use RPCs**: Once registered against a vertical, select the RPC of your choice. Then complete the object fields in the gRPC stub.

To know more about creating a Service Layer API, see. Cisco IOS-XR Service Layer.

Figure 7: Service Layer API Workflow



Note Removing VRF or interface configurations referenced by SL-API objects is not supported and can impact traffic. Ensure Service Layer API clients reroute traffic and update routing before making such changes.

To know more about using gRPC protocol, see Use gRPC Protocol to Define Network Operations with Data Models, on page 45 Chapter in Programmability Configuration Guide.

Preprogram Backup LSPs Using Service Layer API

Table 31: Feature History Table

Feature Name	Release Information	Feature Description
Preprogram Backup LSPs Using Service Layer API	Release 24.2.11	This feature extends the Service Layer API, allowing the controller to preprogram backup Label Switched Paths (LSPs) in the hardware. When the *Path Priority* flag indicates a transition from the backup LSP to the primary LSP, the controller switches the traffic to the backup LSP.

With this feature, the primary LSP failure is detected through a controller-defined mechanism. Upon detecting a failure, the controller switches the primary LSP to backup in a down state and promotes the backup LSP to primary using the provided API parameters.

You can use the Service Layer API to preprogram LSPs as either primary or backup paths by using the *Path Priority* attribute. You can group LSPs with the *set-ID* attribute and determine their operational status as active or inactive using the *Path State* attribute. To ensure seamless traffic management, you can monitor the status of the LSPs using the controller. If traffic needs to be rerouted to the backup LSP, you can modify the priority of the preconfigured backup LSP to primary through the controller, thus allowing the backup path to take over the traffic load. The primary LSP then acts as the backup with its *Path State* set as down to retain the preprogram state. For more information about Service Layer API, see Github - Service Layer API and https://xrdocs.io/cisco-service-layer/.

Verify the Preprogramed Backup Paths

Use the **show service-layer mpls** command to verify the backup programming state for an LSP. For a given path, you can view path priority, and path set ID.

In the following command output, the Next-Hop Label Forwarding Entry 1 (**NHLFE 1**) is the primary LSP as the **path priority** is primary and the LSP state is up. **NHLFE 2** is the backup LSP as the **path priority** is backup and it belongs to the set ID 1. The status of the backup LSP is up.

```
Router#show service-layer mpls label 24000 exp default
Tue Jun 11 04:58:03.154 UTC
vrf name: mpls-default,  vrf state: eof,
vrf magic: valid,  purge timer: 600 seconds,  vrf flags: eof,

local label: 24000, update priority: high, magic: valid, flags: elsp, EXP: default,
    nhlfe: 1, magic: valid,
        ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
        flags: path priority: primary, path setid: 0, path up
        path protection flags: 0, next hop: 10.10.10.2, load metric: 32,
        label action: 1,
        remote address:
        remote labels: 34000,
        interface name: Bundle-Ether1,

    nhlfe: 2, magic: valid,
```

```
    ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
    flags: path priority: backup, path setid: 1, path up
    path protection flags: 0, next hop: 10.10.10.3, load metric: 1,
    label action: 1,
    remote address:
    remote labels: 44000,
    interface name: Bundle-Ether2,

nhlfe: 3, magic: valid,
    ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
    flags: path priority: backup, path setid: 1, path up
    path protection flags: 0, next hop: 10.10.10.8, load metric: 31,
    label action: 1,
    remote address:
    remote labels: 44000,
    interface name: Bundle-Ether3
```

The following table describes the possible values for the path attributes:

Attribute	Possible Values
Path Priority	Primary or Backup
set-ID	0–3
Path State	Up or Down

TPM Enrollment and Attestation

Table 32: Feature History Table

Feature Name	Release Information	Description
TPM Enrollment and Attestation	Release 24.3.1	Introduced in this release on: Fixed Systems (8200, 8700); Centralized Systems (8600); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) You can now use the new gNSI service for enrollment and attestation, EnrollZ and AttestZ, to enhance security of networking devices. The EnrollZ has been added to meet open-source requirements, thereby providing advantages such as the verification of device identity and integrity during boot-up, and the provisioning of owner-specific certificates. This bypasses the need for router vendor certificate authorities, offering a user-friendly and secure system. Sensitive credentials are only available to devices that have completed the EnrollZ and AttestZ processes.

Secure TPM Enrollment and Attestation Workflow for Network Devices

The EnrollZ and AttestZ gNSI services provide a secure method for verifying the identity and integrity of network devices. The EnrollZ service handles the TPM 2.0 enrollment workflow, involving cryptographic verification of the device's TPM-rooted identity and provisioning of attestation and Transport Layer Security (TLS) certificates by the device owner. This ensures that the device is under the control of the owner and not dependent on external vendor Certificate Authorities (CAs) during the attestation process. The AttestZ service manages the TPM 2.0 attestation workflow, confirming the device's integrity throughout the boot process by comparing observed Platform Configuration Register (PCR) values against expected ones to verify the device's boot state. This approach simplifies the TPM enrollment process for device owners, enhances control over certificate management, and eliminates external dependencies, while aligning with Trusted Computing Group (TCG) specifications.

Enroll a TPM 2.0 on Network Devices

The Trusted Platform Module (TPM) 2.0 enrollment workflow is a secure process for network devices to obtain the necessary credentials and configurations for TPM management. This workflow is initiated after the device boot process and involves interaction with various gRPC API endpoints.

Before you begin

- Device has completed the Bootz workflow.
- Device is equipped with a default SSL profile using the Secure Unique Device Identifier (SUDI) key pair and certificate.
- EnrollZ service is available and ready to enroll the TPM on the control card.
- Router owner has access to the trust bundle/anchor from the router vendor.

Step 1 Prepare Device for TPM Enrollment: Ensure the device has completed the Bootz workflow and is ready to serve TPM enrollment gRPC API endpoints on the required port.

Step 2 Trigger EnrollZ Service: Use the `GetIakCert` API to retrieve the Initial Attestation Key (IAK) and IDevID certificates.

Step 3 Verify and Validate Certificates:

- Verify the signature over the IAK certificate using the trust bundle/anchor from the router vendor.
- Confirm that the device identity fields in the IAK and IDevID certificates meet the expected criteria.

Step 4 Request and Install Owner Certificates:

- Request the router owner CA to issue the Owner IAK (oIAK) and Owner IDevID (oIDevID) certificates based on the public keys.
- Use the `RotateOIakCert` API to install the oIAK and oIDevID certificates on the control card.

Step 5 Verify and Store Certificates:

- Verify that the public keys in the oIAK and oIDevID certificates match with respective IAK and SUDI public key.
- Store the oIAK and oIDevID certificates in non-volatile memory for presentation during the TPM attestation (`attestz`) workflow.

Step 6 Update SSL Profile: Update the SSL profile to use the trust bundle and rotate the certificates to the Owner IDevID certificate.

Step 7 Enroll Secondary Control Card: Repeat the enrollment workflow for the secondary control card, if present.

TPM 2.0 Attestation

The TPM 2.0 attestation workflow ensures the integrity and identity of network devices by verifying their configurations and credentials. This process involves interaction with gRPC TPM 2.0 attestation endpoints and requires the device to be booted with the correct OS image and configurations.

Before you begin

- Device must be booted with the correct OS image.
- Correct configurations and credentials must be applied.
- Primary/active control card is responsible for all RPCs directed to the secondary/standby control card.

Step 1 Serve gRPC TPM 2.0 Attestation Endpoints: Ensure the device serves gRPC TPM 2.0 attestation endpoints on port 9339, the same port as gNOI/gNSI/gNMI.

The device must be booted with the correct OS image and configurations.

Step 2 Authenticate Standby Control Card: Perform an authentication handshake between the active and standby control cards using the IDevID key pair/cert.

The active control card is responsible for this handshake as the router owner cannot directly TLS authenticate the standby card.

Step 3 Secure Initial Attestation RPCs: Use the active control card's IDevID private key and oIDevID cert to secure TLS for the initial attestation RPCs.

Step 4 Call AttestZ Service: AttestZ service calls the device's Attest endpoint for a given control card (and a random nonce) to get back:

- An oIAK cert signed by the router owner's CA.
- Final observed PCR hashes/values.
- PCR Quote structure and signature over it signed by IAK private key.
- (Optional) oIDevID cert of the standby control card.

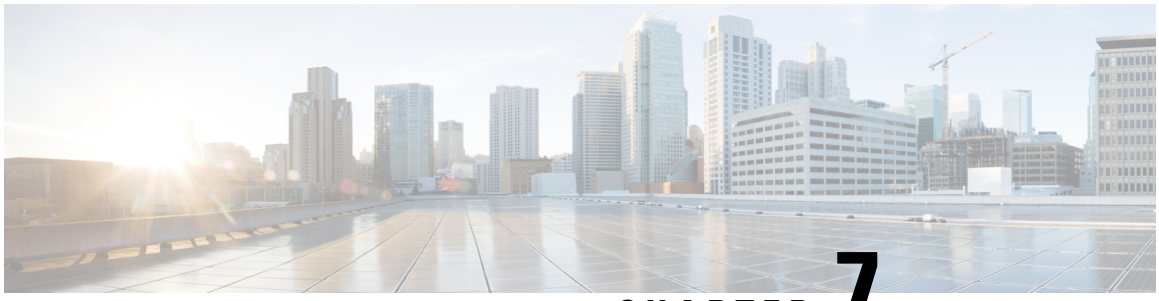
Step 5 Verify Certificates and Signatures:

- AttestZ service uses the trust bundle/anchor from the router owner CA to verify the oIAK cert and its validity/revocation status.
- Ensure that the control card serial number in the oIAK cert and oIDevID cert is the same.

Step 6 Compare PCR Values: The AttestZ service compares the PCR values against the known PCR values provided by the OEM vendor specific to a release.

Step 7 Compare PCR Values and Record Attestation Status: AttestZ service fetches expected final PCR values from its database and compares them to the observed ones reported by the device.

AttestZ service records a successful attestation status for the given control card and repeats the workflow for the secondary/standby control card if one is available.



CHAPTER 7

Enhancements to Data Models

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

Improved YANG Input Validator and Get Requests

Table 33: 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 34: Feature History Table

Feature Name	Release Information	Description
LACP OpenConfig Model	Release 7.5.3	Use the `openconfig-lacp.yang` data model to manage Link Aggregation Control Protocol (LACP) aggregate interfaces by monitoring the number of LACP timeouts and the time since the last timeout. With this release, the data model is revised from version 1.1.0 to 1.2.0 to introduce the following sensor paths for the operational state of the bundle member interface `lacp/interfaces/interface[name]/members/member[interface]/state/`: • `last-change` • `counters/lacp-timeout-transitions` You can stream Event-driven telemetry data for the time since the last change of a timeout, and Model-driven telemetry data for the number of times the state has transitioned with a timeout. The state change is monitored since the time the device restarted or the interface was brought up, whichever is most recent.
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.

Define Power State of Line Card Using Data Model

Table 35: Feature History Table

Feature Name	Release Information	Description
Control Line Card Power Using YANG Data Model	Release 7.5.1	The `oc-platform.yang` YANG data model enables or disables power to the line card and identifies its slot or chassis. You can access this data model from the Github repository.

This feature adds the following component paths to the model to configure and fetch the power state of the line card, enable/disable the power state, and slot ID of line cards:

- /components/component/linecard/config/power-admin-state
- /components/component/linecard/state/power-admin-state
- /components/component/linecard/state/slot-id

```
module: openconfig-platform-linecard
  augment /oc-platform:components/oc-platform:component:
    +--rw linecard
       +--rw config
       |  +--rw power-admin-state?   oc-platform-types:component-power-type
       +--ro state
          +--ro power-admin-state?   oc-platform-types:component-power-type
          +--ro slot-id?             string
```

The following example shows the configuration to enable the line card in location "0/0" to power up:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <edit-config>
  <target>
    <candidate/>
  </target>
  <config>
   <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>0/0</name>
      <linecard xmlns="http://openconfig.net/yang/platform/linecard">
      <config>
       <power-admin-state>POWER_ENABLED</power-admin-state>
      </config>
      </linecard>
     </component>
    </components>
   </config>
 </edit-config>
</rpc>
```

To disable the line card, use `POWER_DISABLED` in the state field.

In the following example, an RPC request is sent to retrieve the power state of all line cards:

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

```
  <filter>
  <components xmlns="http://openconfig.net/yang/platform">
   <component>
    <linecard xmlns="http://openconfig.net/yang/platform/linecard">
     <state/>
    </linecard>
   </component>
  </components>
  </filter>
 </get>
</rpc>
```

The following example shows the RPC response to the request:

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <components xmlns="http://openconfig.net/yang/platform">
   <component>
    <name>0/0</name>
    <linecard xmlns="http://openconfig.net/yang/platform/linecard">
     <state>
      <power-admin-state>POWER_ENABLED</power-admin-state>
      <slot-id>0/0</slot-id>
     </state>
    </linecard>
   </component>
  </components>
 </data>
</rpc-reply>
```

Install Label in oc-platform Data Model

Table 36: Feature History Table

Feature Name	Release Information	Description
Enhancements to openconfig-platform YANG Data Model	Release 7.3.2	The openconfig-platform YANG data model provides a structure for querying hardware and software router components via the NETCONF protocol. This release delivers an enhanced openconfig-platform YANG data model to provide information about: • software version • golden ISO (GISO) label • committed IOS XR packages You can access this data model from the Github repository.

The openconfig-platform (oc-platform.yang) data model is enhanced to provide the following data:

- IOS XR software version (optionally with GISO label)
- Type, description, operational status of the component. For example, a CPU component reports its utilization, temperature or other physical properties.
- List of the committed IOS XR packages

To retrieve oc-platform information from a router via NETCONF, ensure you configured the router with the SH server and management interface:

```
Router#show run
Building configuration...
!! IOS XR Configuration version = 7.3.2
!! Last configuration change at Tue Sep  7 16:18:14 2016 by USER1
!
......
......
netconf-yang agent ssh
ssh server netconf vrf default
interface MgmtEth 0/RP0/CPU0/0
    no shut
    ipv4 address dhcp
```

The following example shows the enhanced `OPERATING_SYSTEM` node component (line card or route processor) of the oc-platform data model:

```
<component>
<name>IOSXR-NODE 0/RP0/CPU0</name>
<config>
<name>0/RP0/CPU0</name>
</config>
<state>
<name>0/RP0/CPU0</name>
<type xmlns:idx="http://openconfig.net/yang/platform-types">idx:OPERATING_SYSTEM</type>
<location>0/RP0/CPU0</location>
<description>IOS XR Operating System</description>
<software-version>7.3.2</software-version> ----------------------> Label Info
<removable>true</removable>
<oper-status xmlns:idx="http://openconfig.net/yang/platform-types">idx:ACTIVE</oper-status>
</state>
<subcomponents>
 <subcomponent>
  <name><platform>-af-ea-7.3.2v1.0.0.1</name>
  <config>
   <name><platform>-af-ea-7.3.2v1.0.0.1</name>
  </config>
  <state>
   <name><platform>-af-ea-7.3.2v1.0.0.1</name>
  </state>
 </subcomponent>
...
```

The following example shows the enhanced `OPERATING_SYSTEM_UPDATE` package component (RPMs) of the oc-platform data model:

```
<component>
<name>IOSXR-PKG/1 <platform>-isis-2.1.0.0-r732</name>
<config>
<name><platform>-isis-2.1.0.0-r732</name>
</config>
<state>
<name><platform>-isis-2.1.0.0-r732</name>
<type xmlns:idx="http://openconfig.net/yang/platform-types">idx:OPERATING_SYSTEM_UPDATE</type>
<description>IOS XR Operating System Update</description>
```

```
<software-version>7.3.2</software-version>----------------------> Label Info
<removable>true</removable>
<oper-status xmlns:idx="http://openconfig.net/yang/platform-types">idx:ACTIVE</oper-status>
</state>
</component>
```

Associated Commands

- **show install committed**—Shows the committed IOS XR packages.
- **show install committed summary**—Shows a summary of the committed packages along with the committed IOS XR version that is displayed as a label.

OpenConfig YANG Model:SR-TE Policies

Table 37: Feature History Table

Feature Name	Release Information	Description
OpenConfig YANG Model:SR-TE Policies	Release 7.3.4	This release supports the OpenConfig (OC) Segment Routing-Traffic Engineering (SR-TE) YANG data model that provides data definitions for SR-TE policy configuration and associated signaling and traffic engineering protocols. Using the model, you can stream a collection of SR-TE operational statistics, such as color, endpoint, and state. You can access the OC data model from the Github repository.

The OC SR-TE policies YANG Data Model supports Version 0.22. Subscribe to the following sensor path to send a pull request to the YANG leaf, list, or container:

`openconfig-network-instance:network-instances/network-instance/segment-routing/te-policies`

The response from the router is a collection of SR-TE operational statistics, such as color, endpoint, and state.

Limitations

- Segment-list ID
  - All locally-configured segment-lists have a unique segment-list ID except for the BGP TE controller. Instead, the BGP TE controller uses the index of the segment-list as the segment-list ID. This ID depends on the local position of the segment-list and can change over time. Therefore for BGP TE controller, you must stream the entire table of the segment-list to ensure that the segment-list ID is always up-to-date.
- Next-hop index
  - The Next-hop container is imported from the `openconfig-aft-common.yang` module where the next-hop index is defined as Uint64. However, the AFT OC in the FIB uses a positional value of

the index and does not identify the next-hop entry separately. Similarly, the next-hop container for OC-SRTE ais also implemented as a positional value of the entry in the list. Ensure that you stream the entire table of the next-hop to get a updated index along with the next-hop entry.

Aggregate Prefix SID Counters for OpenConfig SR YANG Module

Table 38: Feature History Table

Feature Name	Release Information	Description
Aggregate Prefix SID Counters for OpenConfig SR YANG Module	Release 7.3.4	The following components are now available in the OpenConfig (OC) Segment-Routing (SR) YANG model: • The **aggregate-sid-counters** container in the **sr-mpls-top** group to aggregate the prefix segment identifier (SID) counters across the router interfaces. • **The aggregate-sid-counter** and the **mpls-label key** to aggregate counters across all the router interfaces corresponding to traffic forwarded with a particular prefix-SID. You can access the OC data model from the Github repository.

The OpenConfig SR YANG model supports Version 0.3. Subscribe to the following sensor path:

```
openconfig-mpls/mpls/signaling-protocols/segment-routing/aggregate-sid-counters/aggregate-sid-counter/mpls-label/state
```

When a receiver subscribes to the sensor path, the router periodically streams the statistics to telemetry for each SR-label. The default collection interval is 30 seconds.

OpenConfig YANG Model:MACsec

Table 39: Feature History Table

Feature Name	Release Information	Description
OpenConfig YANG Model:MACsec	Release 7.5.2	You can now use the OpenConfig YANG data model to define the MACsec key chain and policy, and apply MACsec encryption on a router interface. You can access the OC data model from the Github repository.

With the OpenConfig YANG Model:MACsec, you can also retrieve operational data from the NETCONF agent using gRPC. By automating processes that are repeated across multiple network elements, you can leverage the YANG models for MACsec.

You can use the following operations to stream Telemetry data by sending a request to the NETCONF agent:

- <get>
- <get-config>
- <edit-config>

Subscribe to the following sensor paths to send a pull request to the YANG leaf, list, or container:

- mka/key-chains/key-chain/mka-keys/mka-key
- interfaces/interface/mka
- interfaces/interface
- mka/policies/policy
- interfaces/interface/scsa-rx/scsa-rx
- interfaces/interface/scsa-tx/scsa-tx
- mka/state/counter

Limitation

- The current implementation of Cisco IOS XR supports only the local time zone configuration in the YYYY-MM-DDTHH:MM:SS format for the following paths:
  - /macsec/mka/key-chains/key-chain/mka-keys/mka-key/config/valid-date-time
  - /macsec/mka/key-chains/key-chain/mka-keys/mka-key/config/expiration-date-time
  - /macsec/mka/key-chains/key-chain/mka-keys/mka-key/state/valid-date-time
  - /macsec/mka/key-chains/key-chain/mka-keys/mka-key/state/expiration-date-time

- Under the MACsec policy, you can disable the delay-protection and include-icv-indicator leaves only by using the delete operation. You cannot modify the configuration by updating the default field value, from true to false. This codeblock shows a sample delete operation:

```
<config>
<delay-protection nc:operation="delete"/>
<include-icv-indicator nc:operation="delete"/>
</config>
```

Running Configuration

```
RP/0/0/CPU0:ios#show running-config
Tue Apr 19 21:36:08.882 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Apr 14 16:25:17 2022 by UNKNOWN
key chain kc
 macsec
  key 1234
   key-string password
00554155500E5D5157701E1D5D4C53404A5A5E577E7E727F6B647040534355560E080A00005B554F4E080A0407070303530A54540C0252445E550958525A771B16
 cryptographic-algorithm aes-256-cmac
   lifetime 00:01:01 january 01 2021 infinite
  netconf-yang agent
 ssh
interface GigabitEthernet0/0/0/0
 shutdown
interface GigabitEthernet0/0/0/1
 macsec psk-keychain kc
interface GigabitEthernet0/0/0/2
 macsec psk-keychain kc policy mp
interface GigabitEthernet0/0/0/3
 shutdown
interface GigabitEthernet0/0/0/4
 shutdown
macsec-policy mp
 cipher-suite GCM-AES-XPN-256
 key-server-priority 4
ssh server v2
end
```

RPC Request for get-config

```
<get-config>
   <source>
      <running/>
   </source>
        <filter>
          <macsec xmlns="http://openconfig.net/yang/macsec">
          </macsec>
        </filter>
</get-config>
```

RPC Response for get-config

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <macsec xmlns="http://openconfig.net/yang/macsec">
   <mka>
    <policies>
     <policy>
```

```
      <name>mp</name>
      <config>
       <name>mp</name>
       <macsec-cipher-suite>gcm-aes-xpn-256</macsec-cipher-suite>
       <key-server-priority>4</key-server-priority>
      </config>
     </policy>
    </policies>
    <key-chains>
     <key-chain>
      <name>kc</name>
      <config>
       <name>kc</name>
      </config>
      <mka-keys>
       <mka-key>
        <id>1234</id>
        <config>
         <id>1234</id>
         <cryptographic-algorithm>AES_256_CMAC</cryptographic-algorithm>
         <valid-date-time>2021-01-01T00:01:01</valid-date-time>
         <expiration-date-time>NO_EXPIRATION</expiration-date-time>
        </config>
       </mka-key>
      </mka-keys>
     </key-chain>
    </key-chains>
   </mka>
   <interfaces>
    <interface>
     <name>GigabitEthernet0/0/0/1</name>
     <config>
      <name>GigabitEthernet0/0/0/1</name>
     </config>
     <mka>
      <config>
       <key-chain>kc</key-chain>
      </config>
     </mka>
    </interface>
    <interface>
     <name>GigabitEthernet0/0/0/2</name>
     <config>
      <name>GigabitEthernet0/0/0/2</name>
     </config>
     <mka>
      <config>
       <key-chain>kc</key-chain>
       <mka-policy>mp</mka-policy>
      </config>
     </mka>
    </interface>
   </interfaces>
  </macsec>
 </data>
</rpc-reply>
```

RPC Request for get

```
<get>
   <filter>
     <macsec xmlns="http://openconfig.net/yang/macsec">
     </macsec>
```

```
    </filter>
</get>
```

RPC Response for get

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <macsec xmlns="http://openconfig.net/yang/macsec">
   <mka>
    <policies>
     <policy>
      <name>mp</name>
      <config>
       <name>mp</name>
       <macsec-cipher-suite>gcm-aes-xpn-256</macsec-cipher-suite>
       <key-server-priority>4</key-server-priority>
      </config>
      <state>
       <name>mp</name>
       <key-server-priority>4</key-server-priority>
       <macsec-cipher-suite>gcm-aes-xpn256</macsec-cipher-suite>
       <confidentiality-offset>zero-bytes</confidentiality-offset>
       <delay-protection>false</delay-protection>
       <include-icv-indicator>false</include-icv-indicator>
       <sak-rekey-interval>0</sak-rekey-interval>
      </state>
     </policy>
     <policy>
      <name>DEFAULT-POLICY</name>
      <state>
       <name>DEFAULT-POLICY</name>
       <key-server-priority>16</key-server-priority>
       <macsec-cipher-suite>gcm-aes-xpn256</macsec-cipher-suite>
       <confidentiality-offset>zero-bytes</confidentiality-offset>
       <delay-protection>false</delay-protection>
       <include-icv-indicator>false</include-icv-indicator>
       <sak-rekey-interval>0</sak-rekey-interval>
      </state>
     </policy>
    </policies>
    <key-chains>
     <key-chain>
      <name>kc</name>
      <config>
       <name>kc</name>
      </config>
      <mka-keys>
       <mka-key>
        <id>1234</id>
        <config>
         <id>1234</id>
         <cryptographic-algorithm>AES_256_CMAC</cryptographic-algorithm>
         <valid-date-time>2021-01-01T00:01:01</valid-date-time>
         <expiration-date-time>NO_EXPIRATION</expiration-date-time>
        </config>
        <state>
         <id>1234</id>
         <cryptographic-algorithm>AES_256_CMAC</cryptographic-algorithm>
         <valid-date-time>2021-01-01T00:01:01</valid-date-time>
         <expiration-date-time>NO_EXPIRATION</expiration-date-time>
        </state>
       </mka-key>
      </mka-keys>
```

```
    <state>
     <name>kc</name>
    </state>
   </key-chain>
  </key-chains>
 </mka>
 <interfaces>
  <interface>
   <name>GigabitEthernet0_0_0_1</name>
   <state>
    <name>GigabitEthernet0_0_0_1</name>
    <counters>
     <tx-untagged-pkts>8</tx-untagged-pkts>
     <rx-untagged-pkts>0</rx-untagged-pkts>
     <rx-badtag-pkts>2</rx-badtag-pkts>
     <rx-unknownsci-pkts>3</rx-unknownsci-pkts>
     <rx-nosci-pkts>4</rx-nosci-pkts>
    </counters>
   </state>
   <mka>
    <state>
     <mka-policy>DEFAULT-POLICY</mka-policy>
     <key-chain>kc</key-chain>
     <counters>
      <in-mkpdu>0</in-mkpdu>
      <in-sak-mkpdu>0</in-sak-mkpdu>
      <out-mkpdu>225271</out-mkpdu>
      <out-sak-mkpdu>0</out-sak-mkpdu>
     </counters>
    </state>
   </mka>
   <scsa-tx>
    <scsa-tx>
     <sci-tx>024f88a08c9d0001</sci-tx>
     <state>
      <sci-tx>024f88a08c9d0001</sci-tx>
      <counters>
       <sc-encrypted>0</sc-encrypted>
       <sa-encrypted>0</sa-encrypted>
      </counters>
     </state>
    </scsa-tx>
   </scsa-tx>
  </interface>
  <interface>
   <name>GigabitEthernet0_0_0_2</name>
   <state>
    <name>GigabitEthernet0_0_0_2</name>
    <counters>
     <tx-untagged-pkts>8</tx-untagged-pkts>
     <rx-untagged-pkts>0</rx-untagged-pkts>
     <rx-badtag-pkts>2</rx-badtag-pkts>
     <rx-unknownsci-pkts>3</rx-unknownsci-pkts>
     <rx-nosci-pkts>4</rx-nosci-pkts>
    </counters>
   </state>
   <mka>
    <state>
     <mka-policy>mp</mka-policy>
     <key-chain>kc</key-chain>
     <counters>
      <in-mkpdu>0</in-mkpdu>
      <in-sak-mkpdu>0</in-sak-mkpdu>
      <out-mkpdu>225271</out-mkpdu>
```

```
          <out-sak-mkpdu>0</out-sak-mkpdu>
         </counters>
        </state>
       </mka>
       <scsa-tx>
        <scsa-tx>
         <sci-tx>0246c822daae0001</sci-tx>
         <state>
          <sci-tx>0246c822daae0001</sci-tx>
          <counters>
           <sc-encrypted>0</sc-encrypted>
           <sa-encrypted>0</sa-encrypted>
          </counters>
         </state>
        </scsa-tx>
       </scsa-tx>
      </interface>
      <interface>
       <name>GigabitEthernet0/0/0/1</name>
       <config>
        <name>GigabitEthernet0/0/0/1</name>
       </config>
       <mka>
        <config>
         <key-chain>kc</key-chain>
        </config>
       </mka>
      </interface>
      <interface>
       <name>GigabitEthernet0/0/0/2</name>
       <config>
        <name>GigabitEthernet0/0/0/2</name>
       </config>
       <mka>
        <config>
         <key-chain>kc</key-chain>
         <mka-policy>mp</mka-policy>
        </config>
       </mka>
      </interface>
     </interfaces>
    </macsec>
   </data>
  </rpc-reply>
```

OpenConfig YANG Model:dscp-set

Table 40: Feature History Table

Feature Name	Release Information	Description
OpenConfig YANG Model:dscp-set	Release 7.5.2	This model allows you to configure a minimum and maximum Differentiated Services Code Point (DSCP) value in the dscp-set leaf-list. When you send these values in your request to the NETCONF agent, it filters the traffic by matching the values in the list with the incoming packet header. This ensures that your network is not vulnerable to unwanted traffic. You can access the OC data model from the Github repository.

You can configure two Differentiated Services Code Point (DSCP) values in the dscp-set leaf-list. You can enter these values in any order, and they are internally mapped to dscp-min and dscp-max values. The incoming IPv4 or IPv6 packet header contains the DSCP field. This DSCP field is matched with the range of values that exist between the specified minimum (dscp-min) and maximum (dscp-max) values. When the DSCP field contains one of the values specified in the list, the incoming packet is allowed access to your network. You can add or delete the dscp-set leaf-list in the IPv4 and IPv6 OpenConfig YANG model by sending a NETCONF request.

Note When you delete one of the values from the dscp-set, the model applies the remaining value for both dscp-min and dscp-max fields.

Adding the dscp-set in the IPv4 OC YANG Model

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
  <target>
   <candidate/>
  </target>
  <config type="subtree"xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acl xmlns="http://openconfig.net/yang/acl">
   <acl-sets>
    <acl-set>
     <name>test-dscp-set</name>
     <type>ACL_IPV4</type>
     <config>
      <name>test-dscp-set</name>
      <type>ACL_IPV4</type>
     </config>
     <acl-entries>
```

```
      <acl-entry>
       <sequence-id>10</sequence-id>
       <config>
        <sequence-id>10</sequence-id>
       </config>
       <actions>
        <config>
         <forwarding-action>ACCEPT</forwarding-action>
        </config>
       </actions>
       <ipv4>
        <config>
         <dscp-set>12</dscp-set>
         <dscp-set>15</dscp-set>
        </config>
       </ipv4>
      </acl-entry>
     </acl-entries>
    </acl-set>
   </acl-sets>
  </acl>
  </config>
</edit-config>
</rpc>
```

Deleting the dscp-set in the IPv4 OC YANG Model

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
  <candidate/>
</target>
<config type="subtree" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <acl xmlns="http://openconfig.net/yang/acl">
  <acl-sets>
    <acl-set xc:operation="delete">
     <name> test-dscp-set</name>
     <type>ACL_IPV4</type>
   </acl-set>
 </acl-sets>
 </acl>
</config>
</edit-config>
</rpc>
```

Adding the dscp-set in the IPv6 OC YANG Model

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
<target>
 <candidate/>
</target>
<config type="subtree" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <acl xmlns="http://openconfig.net/yang/acl">
  <acl-sets>
   <acl-set>
    <name>test-dscp-v6-edit</name>
    <type>ACL_IPV6</type>
    <config>
      <name>test-dscp-v6-edit</name>
      <type>ACL_IPV6</type>
   </config>
   <acl-entries>
```

```
    <acl-entry>
      <sequence-id>10</sequence-id>
       <config>
          <sequence-id>10</sequence-id>
       </config>
      <actions>
      <config>
        <forwarding-action>ACCEPT</forwarding-action>
     </config>
    </actions>
 <ipv6>
  <config>
    <dscp-set>22</dscp-set>
    <dscp-set>55</dscp-set>
  </config>
 </ipv6>
</acl-entry>
</acl-entries>
 </acl-set>
</acl-sets>
</acl>
</config>
</edit-config>
</rpc>
```

Deleting the dscp-set in the IPv6 OC YANG Model

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
  <candidate/>
</target>
<config type="subtree" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <acl xmlns="http://openconfig.net/yang/acl">
  <acl-sets>
    <acl-set xc:operation="delete">
     <name>test-dscp-v6-edit</name>
     <type>ACL_IPV6</type>
   </acl-set>
 </acl-sets>
 </acl>
</config>
</edit-config>
</rpc>
```

OpenConfig YANG Model:procmon

Table 41: Feature History Table

Feature Name	Release Information	Description
OpenConfig YANG Model:procmon	Release 7.5.2	This model provides data definitions to monitor the health of one or more processes running on a system, delivering insights into the performance of critical processes and helping remediate performance bottlenecks. For example, the stress tool that is part of the Linux distribution may be consuming high CPU. The openconfig-procmon model pulls this information and sends it to you when you query the node. As a remediation measure, you can then restart the process. You can access the OC data model from the Github repository.

Subscribe to the following sensor path:

openconfig-system:system/processes/process

Based on a Process ID (PID), you can stream state parameters, such as name, args, start-time, uptime, cpu-usage-user, cpu-usage-system, cpu-utilization, memory usage and memory utilization.

When you send the PID to a MDT-capable device requesting state parameters of a process, the PID of the process acts as a key for the request. If the requested PID is invalid, you will not receive any response.

Note The location of the PID is always assumed to be the Active RP. This model does not have any leaf or field where you can specify the location or node name.

Example

This output shows state parameters that monitor the health of the dhcpd process having PID: 22482 using the XR built-in mdt_exec tool. You can also use telemetry tools, such as gNMI and gRPC.

```
RP/0/RP1/CPU0:SF-D#run mdt_exec -s openconfig-system:system/processes/process[pid=22482]
Enter any key to exit...
 Sub_id 200000001, flag 0, len 0
 Sub_id 200000001, flag 4, len 583
--------
{"node_id_str":"SF-D","subscription_id_str":"app_TEST_200000001",
"encoding_path":"openconfig-system:system/processes/process","collection_id":"13",
"collection_start_time":"1648387172382","msg_timestamp":"1648387172384",
```

```
"data_json":[{"timestamp":"1648387172384","keys":[{"pid":"22482"}],
"content":{"state":{"pid":"22482","name":"dhcpd","args":["dhcpd"],
"start-time":"1648385883000000000","uptime":"1289384179023","cpu-usage-user":"270000000",
"cpu-usage-system":"180000000","cpu-utilization":0,"memory-usage":"16641952",
"memory-utilization":0}}}],"collection_end_time":"1648387172384"}
--------
 Sub_id 200000001, flag 8, len 0
```


Automatic Resynchronization of OpenConfig Configuration

Table 42: 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 43: 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`:

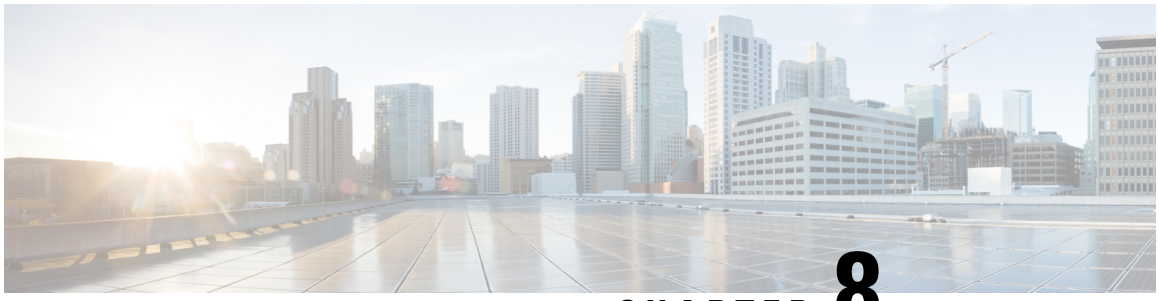
  For a configuration that is out of sync because it failed during a resynchronization attempt, the operation is always `set`, but for a user-initiated commit operation, the operation is whichever the user was attempting during the commit.

- **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 8

Unified Data Models

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

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

You can access the models supported on the router using the following command:

```
Router#run
[node]$cd /pkg/yang
[node:pkg/yang]$ls
```

The unified models are also available in the Github repository.

• Unified Configuration Models, on page 153

Unified Configuration Models

Table 44: 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.
Unified Data Model to Configure checksum in the custom OID	Release 7.5.3	Use the `Cisco-IOS-XR-um-script-cfg.yang` unified data model to configure checksum for the newly added file-name in the Custom OID.
Unified Data Model to Configure Encapsulated Ambiguous VLANs	Release 7.5.3	Use the `Cisco-IOS-XR-um-if-encap-ambiguous-cfg.yang` unified data model to configure encapsulated ambiguous VLANs with IEEE802.1ad Provider Bridging (PB) encapsulation type on an access-interface.

Feature Name	Release Information	Description
Unified Data Model to Configure MAC Address	Release 7.5.3	Use the `Cisco-IOS-XR-um-if-mac-address-cfg.yang` unified data model to set or delete a Media Access Control (MAC) address of the Management Ethernet interface, which acts as a unique identifier for the device in the network.
New Unified Models	Release 7.5.2	Unified models are CLI-based YANG models that are designed to replace the native schema-based models. This release introduces new unified models to configure the Fabric Interface ASIC (FIA), Link Aggregation Control Protocol (LACP), Cisco Express Forwarding (CEF) and controller fabric. You can access these new unified models from the Github repository.
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 45: 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-mac-address-cfg	Release 7.5.3
Cisco-IOS-XR-um-if-encap-ambiguous-cfg	Release 7.5.3
Cisco-IOS-XR-um-cont-cpri-cfg	Release 7.5.2
Cisco-IOS-XR-um-lacp-cfg	Release 7.5.2
Cisco-IOS-XR-um-controller-fabric-cfg	Release 7.5.2
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

Unified Models	Introduced in Release
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
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

Unified Models	Introduced in Release
Cisco-IOS-XR-um-cef-accounting-cfg	Release 7.4.1
Cisco-IOS-XR-um-cfg-mibs-cfg	Release 7.4.1
Cisco-IOS-XR-um-cli-alias-cfg	Release 7.4.1
Cisco-IOS-XR-um-clock-cfg	Release 7.4.1
Cisco-IOS-XR-um-config-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-breakout-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-optics-cfg	Release 7.4.1
Cisco-IOS-XR-um-control-plane-cfg	Release 7.4.1
Cisco-IOS-XR-um-crypto-cfg	Release 7.4.1
Cisco-IOS-XR-um-domain-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-cfm-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-oam-cfg	Release 7.4.1
Cisco-IOS-XR-um-exception-cfg	Release 7.4.1
Cisco-IOS-XR-um-flowspec-cfg	Release 7.4.1
Cisco-IOS-XR-um-frequency-synchronization-cfg	Release 7.4.1
Cisco-IOS-XR-um-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-port-range-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-profile-cfg	Release 7.4.1
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

Unified Models	Introduced in Release
Cisco-IOS-XR-um-logging-correlator-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-domain-cfg	Release 7.4.1
Cisco-IOS-XR-um-lpts-pifib-dynamic-flows-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-cbqosmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-fabric-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-ifmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-rfmib-cfg	Release 7.4.1
Cisco-IOS-XR-um-mibs-sensormib-cfg	Release 7.4.1
Cisco-IOS-XR-um-monitor-session-cfg	Release 7.4.1
Cisco-IOS-XR-um-mpls-oam-cfg	Release 7.4.1
Cisco-IOS-XR-um-ntp-cfg	Release 7.4.1
Cisco-IOS-XR-um-pce-cfg	Release 7.4.1
Cisco-IOS-XR-um-pool-cfg	Release 7.4.1
Cisco-IOS-XR-um-priority-flow-control-cfg	Release 7.4.1
Cisco-IOS-XR-um-rcc-cfg	Release 7.4.1
Cisco-IOS-XR-um-router-hsrp-cfg	Release 7.4.1
Cisco-IOS-XR-um-router-vrrp-cfg	Release 7.4.1
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

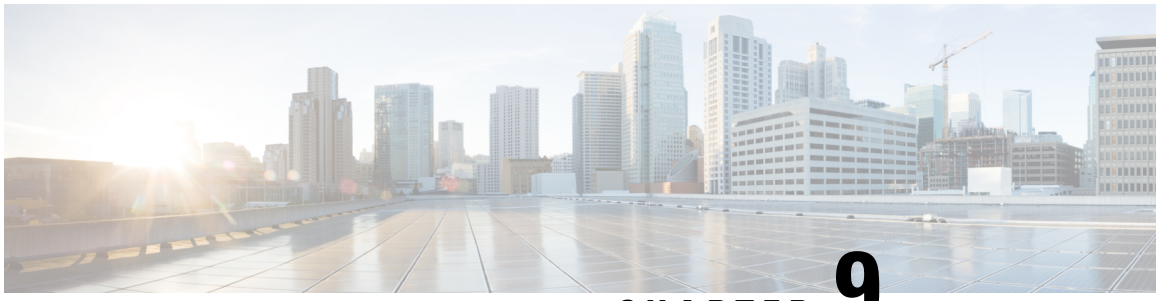
Unified Models	Introduced in Release
Cisco-IOS-XR-um-traps-flash-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-fru-ctrl-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-ipsec-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-l2tun-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-otn-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-power-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-selective-vrf-download-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-syslog-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-system-cfg	Release 7.4.1
Cisco-IOS-XR-um-udp-cfg	Release 7.4.1
Cisco-IOS-XR-um-vty-pool-cfg	Release 7.4.1
Cisco-IOS-XR-um-xml-agent-cfg	Release 7.4.1
Cisco-IOS-XR-um-conflict-policy-cfg	Release 7.3.1
Cisco-IOS-XR-um-flow-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-access-group-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv4-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv6-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-service-policy-qos-cfg	Release 7.2.1
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

Unified Models	Introduced in Release
Cisco-IOS-XR-um-if-l2transport-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv4-prefix-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv6-prefix-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-amt-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-mld-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-msdp-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-bgp-cfg	Release 7.1.1
Cisco-IOS-XR-um-mpls-te-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-isis-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-ospf-cfg	Release 7.1.1
Cisco-IOS-XR-um-router-ospfv3-cfg	Release 7.1.1
Cisco-IOS-XR-um-grpc-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-bundle-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-ethernet-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-ip-address-cfg	Release 7.0.1
Cisco-IOS-XR-um-if-vrf-cfg	Release 7.0.1
Cisco-IOS-XR-um-interface-cfg	Release 7.0.1
Cisco-IOS-XR-um-mpls-l3vpn-cfg	Release 7.0.1
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

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

PART II

Automation Scripts



CHAPTER 9

Achieve Network Operational Simplicity Using Automation Scripts

Table 46: Feature History Table

Feature Name	Release Information	Description
Operational Simplicity Using Automation Scripts	Release 7.3.2	This feature lets you host and execute your automation scripts directly on a router running IOS XR software, instead of managing them on external controllers. The scripts available on-box can now leverage Python libraries, access the underlying router information to execute CLI commands, and monitor router configurations continuously. This results in setting up a seamless automation workflow by improving connectivity, access to resources, and speed of script execution. The following categories of on-box scripts are used to achieve operational simplicity:

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, on page 164

Explore the Types of Automation Scripts

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

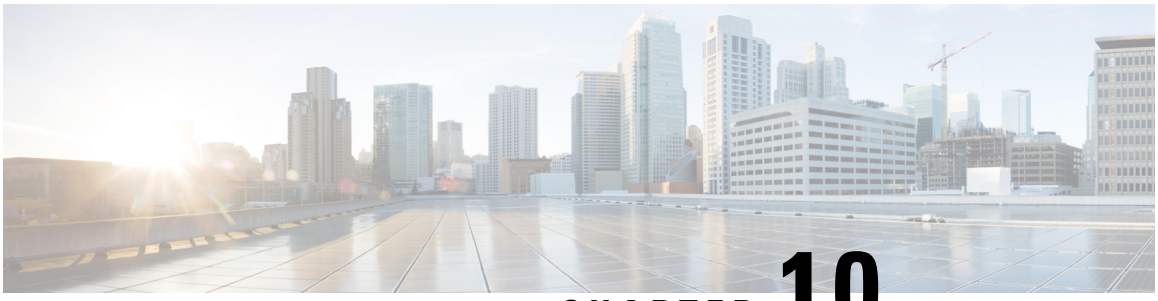
- Configuration (Config) scripts
- Execution (Exec) scripts
- Process scripts
- EEM scripts

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

Table 47: 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.

	Config Scripts	Exec Scripts	Process Scripts	EEM Scripts
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 10

Precommit Scripts

Table 48: Feature History Table

Feature Name	Release Information	Description
Precommit Script to Validate Configuration Change	Release 7.5.4	With this feature, you can deploy custom python scripts to be executed automatically during a configuration commit operation. These scripts process the configuration change and act as deciding factor to either proceed with applying the configuration or stop the commit operation in the event of an error.

Cisco IOS XR precommit scripts can validate the configuration during the commit operation. They allow device administrators to enforce custom configuration validation rules. These scripts are invoked automatically when you change a configuration and commit the changes. When a configuration commit is in progress, a precommit script is automatically initaited to validate the changes. If the change is valid, the script allows committing the new configuration. If the configuration is invalid, or does not adhere to the enforced validation rules, the script notifies you about the mismatch and blocks the commit operation. Overall, precommit scripts help to maintain crucial device parameters, and reduce human error in managing the network.

When you commit a configuration, the system automatically invokes the precommit scripts to validate that change. Precommit scripts can perform the following actions during a commit operation:

- Validate the proposed new configuration, ensure that the changes to the target configuration does not exceed the boundaries defined for the system or software functionality. For example, you can program the script to estimate the Ternary Content Addressable Memory (TCAM) slots needed for the target configuration, and verify that the TCAM usage does not exceed a defined threshold.
- Verify that the commit operation adheres to the predefined execution rules. For example, you can use the script to ensure that certain configuration changes that impact traffic are allowed only at specified time intervals.
- Block the commit operation if the configuration is invalid and notify the details in an error message.
- Generate system log messages for in-depth analysis of the configuration change. This log also helps in troubleshooting a failed commit operation.

Precommit Script Limitations

The following restrictions apply when using precommit scripts:

- Precommit scripts cannot modify a configuration.
- Configuration validation before a commit operation is supported only using CLI commands. Operations using NETCONF, gNMI and XML are not supported even if the precommit script is enabled.

Get Started with Precommit Scripts

Precommit scripts can be written in Python 3.9 (and earlier) programming language using the packages that Cisco supports. For more information about the supported packages, see Script Infrastructure and Sample Templates, on page 265.

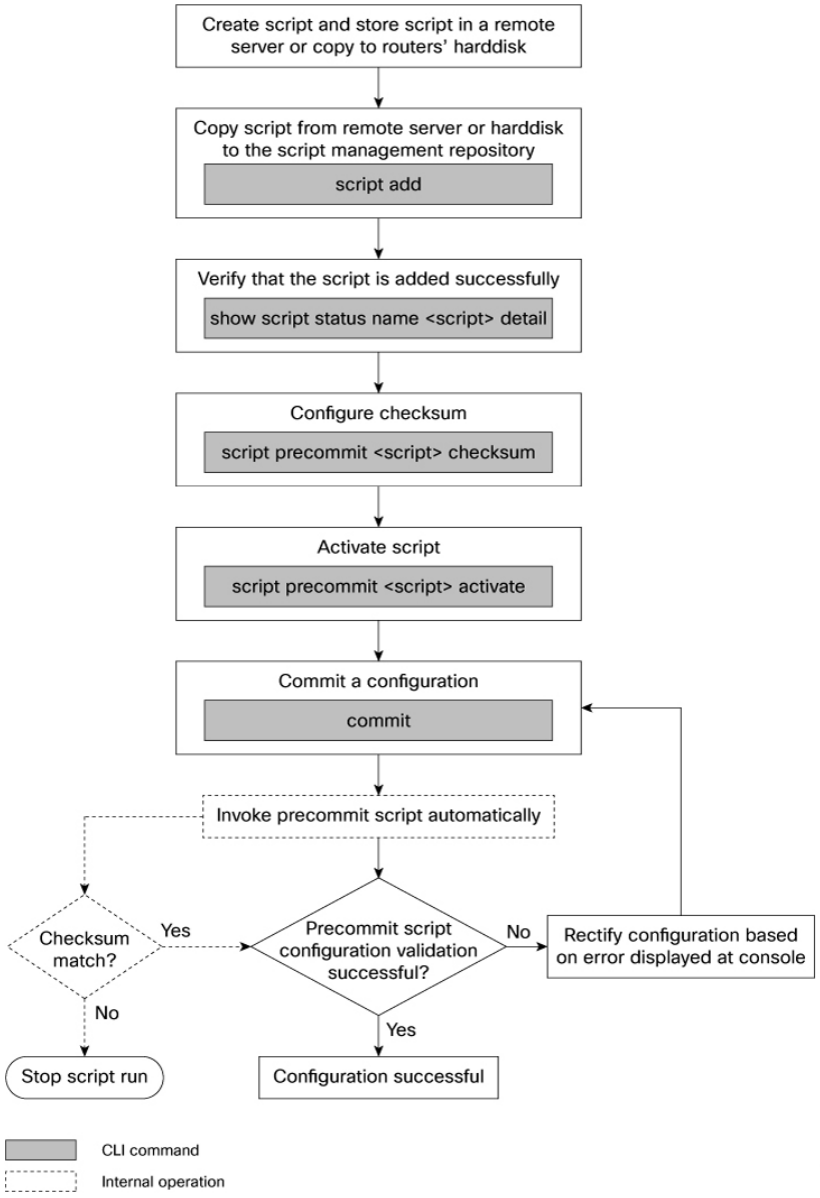
This chapter gets you started with provisioning your precommit automation scripts on the router.

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

- Workflow to Run Precommit Scripts, on page 168
- Example: Verify BGP Configuration Using Precommit Script, on page 174

Workflow to Run Precommit Scripts

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



Complete the following tasks to provision precommit scripts:

- Download the Script to the Router—Store the precomit script on a remote server or copy to the harddisk of the router. Add the precommit script from the server to the script management repository (`hardisk:/mirror/script-mgmt`) on the router using the **script add precommit** command.
- Configure Checksum for Precommit Script—Configure the script integrity and authenticity using the **script precommit** *script* **checksum** command. A script cannot be used unless the checksum is configured.
- Activate Precommit Scripts—Activate the precommit script using **script precommit** *script* **activate** command to validate the configuration from a commit operation. The script ensures that the configuration changes comply with the predefined conditions in the script, and uncover potential errors, if any.

Note A precommit script is invoked automatically when you commit a configuration change to modify the router configuration. You can view the result from the script execution on the console.

Download the Script to the Router

Script Type	Download Location
precommit	`harddisk:/mirror/script-mgmt/precommit`
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 `precommit-bgp.py` script to the script management repository.

Before you begin

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.

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 remote server (HTTP, HTTPS, FTP or SCP) or the harddisk location in the router.

  ```
  Router#script add precommit script-location script.py
  ```

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

  ```
  Router#script add precommit http://192.0.2.0/scripts precommit-bgp.py
  Tue Jan 24 05:03:40.791 UTC
  Copying script from http://192.0.2.0/scripts/precommit-bgp.py
  precommit-bgp.py has been added to the script repository
  ```

  You can add a maximum of 10 scripts simultaneously.

  ```
  Router#script add precommit 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.

Note Only SHA256 checksum is supported.

```
Router#script add precommit http://192.0.2.0/scripts precommit-bgp.py checksum SHA256 checksum-value
```

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

```
Router#script add precommit 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.

- **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/precommit-bgp.py /harddisk:/
  ```

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

  ```
  Router#script add precommit /harddisk:/ precommit-bgp.py
  Tue Jan 24 05:03:40.791 UTC
  Copying script from /harddisk:/precommit-bgp.py
  precommit-bgp.py has been added to the script repository
  ```

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

Example:

```
Router#show script status
Tue Jan 24 05:10:40.791 UTCC
=============================================================================================================

 Name                      | Type        | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 precommit-bgp.py          | precommit   | Config Checksum  | NEW         | Tue Jan 24 05:10:18 2023
```

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

Configure Checksum for Precommit Script

Every script is associated with a checksum hash value. This value ensures the integrity of the script, and that the script is not tampered with. The checksum is a string of numbers and letters that act as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a syslog warning message is displayed.

It is mandatory to configure the checksum to run the script.

Note Precommit scripts support SHA256 checksum.

Step 1 Retrieve the SHA256 checksum hash value for the script. Ideally this action would be performed on a trusted device, such as the system on which the script was created. This minimizes the possibility that the script is tampered with. However, if the router is secure, you can retrieve the checksum hash value from the IOS XR Linux bash shell.

Example:

```
Router#run
[node0_RP0_CPU0:~]$sha256sum /harddisk:/mirror/script-mgmt/precommit/precommit-bgp.py
6bb460920a694a0f91a27892f457203090e7a6391ab7d2f8656f477af17f9ed1
/harddisk:/mirror/script-mgmt/precommit/precommit-bgp.py
```

Make note of the checksum value.

Step 2 View the status of the script.

Example:

```
Router#show script status detail
Tue Jan 24 05:20:13.539 UTC
=============================================================================================================

 Name                       | Type         | Status           | Last Action    | Action Time
-------------------------------------------------------------------------------------------------------------

 precommit-bgp.py           | precommit    | Config Checksum  | NEW            | Tue Jan 24 05:19:41
2023
-------------------------------------------------------------------------------------------------------------

 Script Name       : precommit-bgp-script.py
 History:
 --------
 1.   Action       : NEW
      Time         : Tue Jan 24 05:19:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The `Status` shows that the checksum is not configured.

You can view the details of the specific script using the **show script status name** *script* **detail** command.

Step 3 Configure the checksum and set the priority.

Example:

```
Router#configure
Router(config)#script precommit precommit-bgp.py checksum SHA256
6bb460920a694a0f91a27892f457203090e7a6391ab7d2f8656f477af17f9ed1 priority 20
Router(config)#commit
Tue Jan 24 10:23:10.546 UTC
Router(config)#end
```

If you are configuring multiple scripts, the system decides an appropriate order to run the scripts. However, you can control the order in which scripts execute using a priority value. For more information on configuring the priority value, see Control Priority When Running Multiple Scripts, on page 189.

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
Tue Jan 24 05:06:17.296 UTC
=============================================================================================================

 Name                          | Type       | Status            | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 precommit-bgp.py              | precommit | Ready             | NEW         | Tue Jan 24 06:17:41 2023
-------------------------------------------------------------------------------------------------------------

 Script Name        : precommit-bgp.py
 Checksum           : 6bb460920a694a0f91a27892f457203090e7a6391ab7d2f8656f477af17f9ed1
 History:
 --------
 1.   Action        : NEW
      Time          : Tue Jan 24 06:17:41 2023
      Checksum      : 6bb460920a694a0f91a27892f457203090e7a6391ab7d2f8656f477af17f9ed1
      Description   : User action IN_CLOSE_WRITE
=============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script is not run, and the commit operation that triggered the script is rejected. It is mandatory for the checksum values to match for the script to run.

Activate Precommit Scripts

Activate the precommit script to validate a configuration change on the set of active configuration (including any scripts newly activated as part of the configuration change) before committing the changes.

Note If the precommit script rejects one or more items in the configuration change, the entire configuration is rejected before committing the change.

Before you begin

Ensure that the following prerequisites are met before you run the script:

1. Download the Script to the Router, on page 170
2. Configure Checksum for Precommit Script, on page 171

Step 1 Activate the precommit script for the configuration validation to take effect.

Example:

```
Router(config)#script precommit precommit-bgp.py activate
```

Step 2 Commit the changes and verify that the precommit script is automatically initiated. You can choose to perform one of the following options based on the requirement:

- Commit the changes to automatically initiate the precommit verification script.

```
Router(config-bgp-nbr)#commit
Tue Jan 24 00:13:37.050 UTC
Precommit Script Report Start
-----------------------------
Pre-commit Verification Result: Pass
Pre-commit Verification Script precommit-bgp.py (req id 1656378102): Pass
-----------------------------
Precommit Script Report Done
```

- Ignore the result of the precommit script execution and proceed to the next step in the commit process using **ignore-results** keyword. Use this keyword if you want to bypass the commit verification. The precommit script is still executed, but the result is ignored.

  ```
  Router(config-bgp-nbr)#commit script-verification ignore-results
  ```

- View all the logs generated by the commit script on the console using **verbose** keyword. If this keyword is not specified, only the result of the script verification is displayed on the console.

  ```
  Router(config-bgp-nbr)#commit script-verification verbose
  ```

An execution report from the script is displayed on the console. If the script displays an error message, rectify the error and rerun the commit operation. If there are no validation errors, the commit operation is successful indicating that the configuration change is valid.

Example: Verify BGP Configuration Using Precommit Script

In this example, you create a precommit script to validate the following Border Gateway Protocol (BGP) configuration:

- Check that the autonomous system (AS) value is in the range from `123` to `234`
- Check that the remote AS of neighbours is not set to `25`

Step 1 Create a precommit script named `verify-bgp.py`. Store the script on a remote server or copy the script to the harddisk: location of the router.

Example:

```
"""

import re
from iosxr.xrcli.xrcli_helper import XrcliHelper
from cisco.script_mgmt import xrlog
from cisco.script_mgmt import precommit

syslog = xrlog.getSysLogger('precommit_verify_bgp')
log = xrlog.getScriptLogger('precommit_verify_bgp')
helper = XrcliHelper(debug=True)


def verify_bgp():
    """
    Query for target configs and check if the target configs has bgp configs
    Check if the bgp AS is in the range 123-234
    Check if remote AS is not 25.
    :return: None on pass / Raise exception on failure.
```

```
    """

    # CLI verification
    cfg = precommit.get_target_configs()
    #cfg = "Thu Feb 23 18:54:28.605 UTC\nrouter bgp 100\n neighbor 10.0.0.1\n remote-as 25\n !\n!\n"

    #cfg = cfg.split("\n")
    print(cfg)

    for cfg_line in cfg:

        bgp_cfg_start_pattern = re.match("^router bgp (.*)", cfg_line)
        if bgp_cfg_start_pattern:
            log.info("BGP config found")

            bgp_as = int(bgp_cfg_start_pattern.group(1))
            if not bgp_as in range(123, 234):
                precommit.config_warning("BGP AS number (%d) " % bgp_as +
                                         "not in recommended range (123-234)")

    # sysdb verification
    cfg = precommit.get_target_configs(format="sysdb")
    # cfg = [Item(name='gl/ip-bgp/default/0/100/aya', value=1, datatype=1),
    # Item(name='gl/ip-bgp/default/0/100/gbl/edm/ord_a/running', value=1, datatype=1),
    #
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/________/________/aya',
 value=1, datatype=1),
    #
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/________/________/ord_a/exists',
 value=1, datatype=1),
    #
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/________/________/ord_b/remote-as',
 value=(0, 26), datatype=5)]
    print(cfg)

    for item in cfg:

        remote_as_pattern = re.match("^gl/ip-bgp/default/0/.*/remote\-as", item.name)
        if remote_as_pattern:
            log.info("BGP remote AS config found")
            remote_as = int(item.value[1])
            if remote_as == 25:
                syslog.info("Attempt to configure BGP remote AS %d" % remote_as)
                precommit.config_error("Remote AS (%d) is not permitted" % remote_as)

    log.info("BGP verification is good")


if __name__ == '__main__':

    result = helper.xrcli_exec("show version")
    match = re.search(r'Version +: (.*)\n*', result['output'])
    print("Image version: %s" % match.group(1))
    verify_bgp()
```

Step 2 Add the script from the remote server or the harddisk: location to the script management repository. See Download the Script to the Router, on page 170.

Step 3 Configure the checksum value to check the script integrity. See Configure Checksum for Precommit Script, on page 171.

Step 4 Activate the script. See Activate Precommit Scripts, on page 173.

Step 5 Configure BGP and commit the configuration.

Example:

```
Router(config)#router bgp 100
Router(config-bgp)#neighbor 10.0.0.1
Router(config-bgp-nbr)#remote-as 25
Router(config-bgp-nbr)#commit
Wed Jan 25 22:53:21.910 UTC
Precommit Script Report Start
----------------------------
Pre-commit Verification Result: Fail
Pre-commit Verification Script verify-bgp.py (req id 1674671641):  Fail
% Script exception return value 1
Errors:
  Remote AS (25) is not permitted
 Warnings:
  BGP AS number (100) not in recommended range (123-234)
----------------------------
Precommit Script Report Done

% Failed to commit .. As an error (Unknown) encountered during commit operation. Changes may not have
 been committed:
'SCRIPT_MGMT' detected the 'fatal' condition 'One or more Pre-Commit script verifications failed'
```

The precommit script is automatically initiated when you commit the configuration. The result from the script run is displayed.

In this example, the precommit script validates the BGP configuration. The AS value limit that is configured in the script is not within the permissible range of 123 to 234. The script rejects the configuration, and displays the details of the validation failure on the console.

Step 6 Verify the script execution details. You can either choose to ignore the script results or view the detailed report of the script execution.

- Ignore the script results using **ignore-results** keyword, and proceed to commit the configuration.

```
Router(config-bgp-nbr)#commit script-verification ignore-results
Wed Jan 25 23:00:02.057 UTC
Precommit Script Report Start
----------------------------
Pre-commit Verification Result: Pass (Failures Ignored)
Pre-commit Verification Script verify-bgp.py (req id 1674671645):  Fail (Ignored)
% Script exception return value 1
Errors:
  Remote AS (25) is not permitted
 Warnings:
  BGP AS number (100) not in recommended range (123-234)
----------------------------
Precommit Script Report Done
```

- View the detailed report using **verbose** keyword.

```
Router(config-bgp-nbr)#commit script-verification verbose
Wed Jan 25 22:53:30.881 UTC
Precommit Script Report Start
----------------------------
Pre-commit Verification Result: Fail
Pre-commit Verification Script verify-bgp.py (req id 1674671642):  Fail
% Script exception return value 1
Errors:
  Remote AS (25) is not permitted
 Warnings:
  BGP AS number (100) not in recommended range (123-234)
 Script output logs:
/harddisk:/mirror/script-mgmt/logs/verify-bgp.py_precommit_1674671642/stdout.log
Image version: 7.5.4.29I
['!! IOS XR Configuration 7.5.4.29I', 'router bgp 100', ' neighbor 10.0.0.1', '  remote-as 25',
```

```
 ' !', '!', 'end', '', '']
[2023-01-25 22:53:31,545] INFO [precommit_verify_bgp]:: BGP config found
!!!!!$$$$$CONFIG WARNING: BGP AS number (100) not in recommended range (123-234) $$$$$!!!!!
[Item(name='gl/ip-bgp/default/0/100/aya', value=1, datatype=1),
Item(name='gl/ip-bgp/default/0/100/gbl/edm/ord_a/running',
value=1, datatype=1),
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/________/________/aya',
 value=1,
datatype=1),
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/________/________/
ord_a/exists', value=1, datatype=1),
Item(name='gl/ip-bgp/default/0/100/ord_a/default/nbr/________/edm/ord_u/0x3/10.0.0.1/
________/________/ord_b/remote-as', value=(0, 25), datatype=5)]
[2023-01-25 22:53:31,571] INFO [precommit_verify_bgp]:: BGP remote AS config found
!!!!!$$$$$CONFIG ERROR: Remote AS (25) is not permitted $$$$$!!!!!

Script error logs: /harddisk:/mirror/script-mgmt/logs/verify-bgp.py_precommit_1674671642/stderr.log
Traceback (most recent call last):
  File "/harddisk:/mirror/script-mgmt/precommit/verify-bgp.py", line 107, in <module>
    verify_bgp()
  File "/harddisk:/mirror/script-mgmt/precommit/verify_bgp.py", line 97, in verify_bgp
    precommit.config_error("Remote AS (%d) is not permitted" % remote_as)
  File "infra/script-mgmt/src/Packages/precommit.py", line 87, in config_error
cisco.script_mgmt.precommit.PrecommitConfigError: !!!!!$$$$$CONFIG ERROR: Remote AS (25) is not
 permitted $$$$$!!!!!

-----------------------------
Precommit Script Report Done

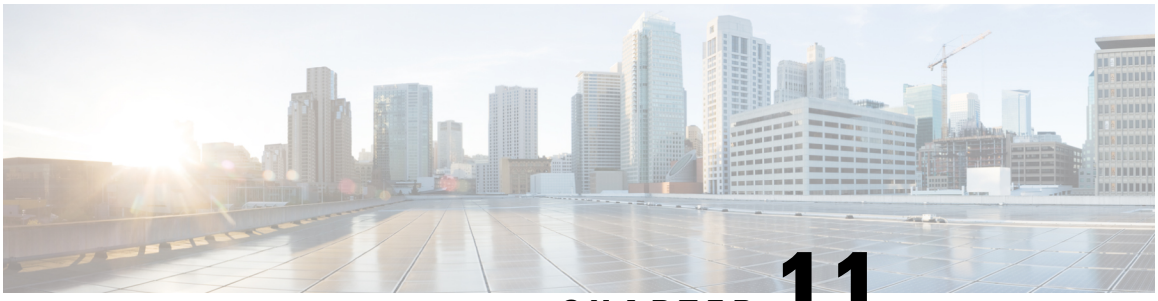
% Failed to commit .. As an error (Unknown) encountered during commit operation. Changes may not
 have been committed:
'SCRIPT_MGMT' detected the 'fatal' condition 'One or more Pre-Commit script verifications failed'
```

Step 7 Rectify the errors and commit the configuration.

Example:

```
Router(config)#router bgp 200
Router(config-bgp)#neighbor 10.0.0.1
Router(config-bgp-nbr)#remote-as 26
Router(config-bgp-nbr)#commit
Wed Jan 25 22:59:06.704 UTC
Precommit Script Report Start
-----------------------------
Pre-commit Verification Result: Pass
Pre-commit Verification Script verify-bgp.py (req id 1674671644):  Pass
-----------------------------
Precommit Script Report Done
```

The precommit script validates the BGP configuration to ensure that the conditions configured in the script are met.



CHAPTER 11

Config Scripts

Cisco IOS XR config scripts can validate and make modifications to configuration changes. They allow device administrators to enforce custom configuration validation rules, or to simplify certain repetitive configuration tasks. These scripts are invoked automatically when you change a configuration and commit the changes. When a configuration commit is in progress, a config script inserts itself into the commit process. The config script can modify the current config candidate. For example, consider you want to maintain certain parameters for routers such as switched off ports or security policies. The config script is triggered to validate the updated configuration and take appropriate action. If the change is valid, the script allows committing the new configuration. If the configuration is invalid, or does not adhere to the enforced constraints, the script notifies you about the mismatch and blocks the commit operation. Overall, config scripts help to maintain crucial device parameters, and reduce human error in managing the network.

When you commit or validate a configuration change, the system invokes each of the active scripts to validate that change. Config scripts can perform the following actions:

- Analyze the proposed new configuration.
- If the configuration is invalid, block the commit by returning an error message along with the set of configuration items to which it relates.
- Return a warning message with the related details but does not block the commit operation.
- Modify the configuration to be included in the commit operation to make the configuration valid, or to simplify certain repetitive configuration tasks. For example, where a value needs duplicating between one configuration item and another configuration item.
- Generate system log messages for in-depth analysis of the configuration change. This log also helps in troubleshooting a failed commit operation.

Config Scripts Limitations

The following are the configuration and software restrictions when using config scripts:

- Config scripts cannot make modifications to configuration that is protected by CCV process, in particular:
  - Script checksum configuration.
  - Other sensitive security configuration such as AAA configuration.
- Config scripts do not explicitly support importing helper modules or other custom imports to provide shared functionality. Although such imports appear to function correctly when set up, they can potentially represent a security risk becaue there is no checksum validation on the imported modules. Modifications

to these imported modules are not automatically detected. To reflect changes to the imported module in the running scripts, you must manually unconfigure and reconfigure any scripts using the imported module.

Get Started with Config Scripts

Config scripts can be written in Python 3.5 programming language using the packages that Cisco supports. For more information about the supported packages

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

Note This chapter 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 Config Scripts, on page 180
- Manage Scripts, on page 188
- Example: Validate and Activate an SSH Config Script, on page 190

Workflow to Run Config Scripts

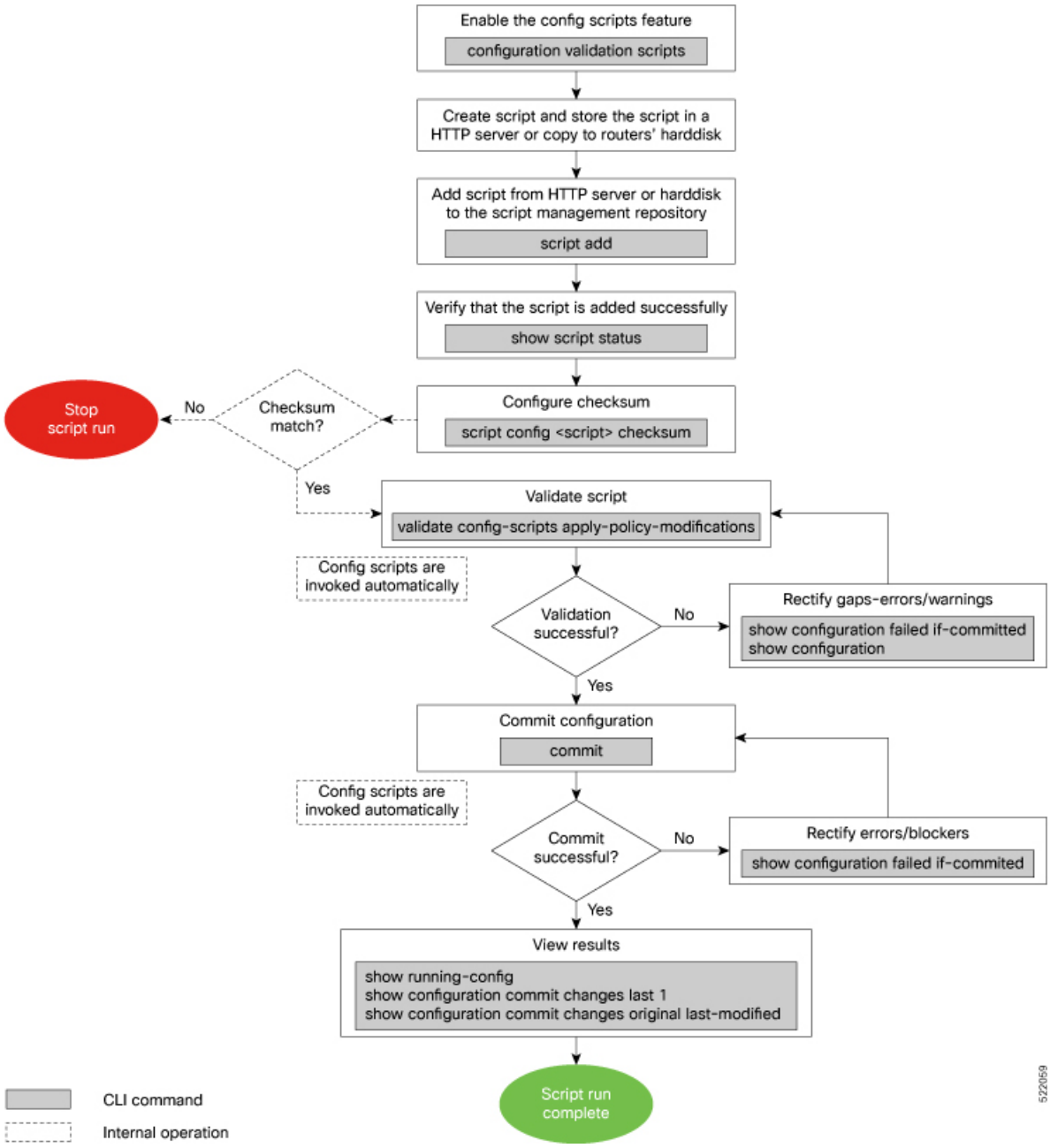
Complete the following tasks to provision config scripts:

- Enable the config scripts feature—Globally activate the config scripts feature on the router using **configuration validation scripts** command.
- Download the script—Store the config script on an HTTP server or copy to the harddisk of the router. Add the config script from the HTTP server to the script management repository (`hardisk:/mirror/script-mgmt`) on the router using the **script add config** command.
- Validate the script—Check script integrity and authenticity using the **script config** *script.py* **checksum** command. A script cannot be used unless the checksum is configured. After the checksum is configured, the script is active.

Note A config script is invoked automatically when you validate or commit a configuration change to modify the candidate configuration.

- Validate the configuration—Ensure that the configuration changes comply with the predefined conditions in the script and uncover potential errors using **validate config-scripts apply-policy-modifications** command.
- View the script execution details—Retrieve the operational data using the **show operational Config Global Validation Script Execution** command.

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



Enable Config Scripts Feature

Config scripts are driven by commit operations. To run the config scripts, you must enable the feature on the router. You must have root user privileges to enable the config scripts.

Note You must commit the configuration to enable the config scripts feature before committing any script checksum configuration.

Step 1 Enable the config scripts.

Example:

```
Router(config)#configuration validation scripts
```

Step 2 Commit the configuration.

Example:

```
Router(config)#commit
```

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 `config-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 config <script-location> <script.py>
  ```

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

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

  You can add a maximum of 10 scripts simultaneously.

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

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

```
Router#script add config 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/config-script.py /harddisk:/
  ```

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

  ```
  Router#script add config /harddisk:/ config-script.py
  Fri Aug 20 05:03:40.791 UTC
  config-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
Fri Sep  2 21:37:05.021 PDT
=============================================================================================================

 Name                             | Type    | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 CpuCheck_Netconf_RPC_Agent.py    | process| Ready            | NEW         | Fri Sep  2 20:24:58 2022

 config_ssh_script.py            | config | Ready            | MODIFY      | Tue Aug 30 14:11:25 2022

 eem_script_action_gshut.py      | eem    | N/A              | MODIFY      | Thu Sep  1 14:37:58 2022
 :23 2021


Router# show appmgr process-script CpuCheck_Netconf_RPC_Agent_Process_App info
Fri Sep  2 21:38:27.455 PDT
Application: CpuCheck_Netconf_RPC_Agent_Process_App
  Activated configuration:
    Executable               : CpuCheck_Netconf_RPC_Agent.py
    Run arguments            : 15
    Restart policy           : On Failure
    Maximum restarts         : 3
  Execution status and info:
    Activated                : Yes
    Status                   : Started
    Executable Checksum      : ee3c32a7d95b398a7eeea9b0d39d4d414338cc9fca739462b8ed49069d28d83c
    Restart count            : 2
    Log location             :
```

```
/harddisk:/mirror/script-mgmt/logs/CpuCheck_Netconf_RPC_Agent.py_process_CpuCheck_Netconf_RPC_Agent_Process_App

    Last started Time        : Fri Sep  2 21:13:33 2022
```

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

Configure Checksum for Config Script

Every script is associated with a checksum hash value. This value ensures the integrity of the script, and that the script is not tampered with. The checksum is a string of numbers and letters that act as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a syslog warning message is displayed.

It is mandatory to configure the checksum to run the script.

Note Config scripts support SHA256 checksum.

Before you begin

Ensure that the following prerequisites are met before you run the script:

1. Enable Config Scripts Feature, on page 181
2.

Step 1 Retrieve the SHA256 checksum hash value for the script. Ideally this action would be performed on a trusted device, such as the system on which the script was created. This minimizes the possibility that the script is tampered with. However, if the router is secure, you can retrieve the checksum hash value from the IOS XR Linux bash shell.

Example:

```
Router#run
[node0_RP0_CPU0:~]$sha256sum /harddisk:/mirror/script-mgmt/config/config-script.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
/harddisk:/mirror/script-mgmt/config/config-script.py
```

Make note of the checksum value.

Step 2 View the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:04:13.539 UTC
=============================================================================================================

 Name                            | Type    | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 config-script.py                | config | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : config-script.py
 History:
 --------
```

```
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
==============================================================================================
```

The `Status` shows that the checksum is not configured.

Step 3 Configure the checksum.

Example:

```
Router#configure
Router(config)#script config config-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Tue Aug 24 10:23:10.546 UTC
Router(config)#end
```

Note When you commit this configuration, the script is automatically run to validate the resulting running configuration. If the script returns any errors, this commit operation fails. This way, the running configuration always remains valid with respect to all currently active scripts with checksums configured.

If you are configuring multiple scripts, the system decides an appropriate order to run the scripts. However, you can control the order in which scripts execute using a priority value. For more information on configuring the priority value, see Control Priority When Running Multiple Scripts, on page 189.

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:06:17.296 UTC
=============================================================================================================

 Name                            | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 config-script.py                | config | Ready            | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : config-script.py
 Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Checksum     : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
      Description  : User action IN_CLOSE_WRITE
==============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script is not run, and the commit operation that triggered the script is rejected. It is mandatory for the checksum values to match for the script to run.

Validate or Commit Configuration to Invoke Config Script

Table 49: Feature History Table

Feature Name	Release Information	Description
Validate Pre-configuration Using Config Scripts	Release 7.5.1	This feature allows you to use config scripts to validate pre-configuration during a commit or validate operation. Any active config scripts can read and validate (accept, reject or modify) pre-configuration. The pre-configuration is only applied to the system later on, when the relevant hardware is inserted, and does not require further script validation at that point. Previously, config scripts did not allow validating configuration until the corresponding hardware was present.

You can validate a configuration change on the set of active config scripts (including any scripts newly activated as part of the configuration change) before committing the changes. This validation ensures that the configuration complies with predefined conditions defined in the active scripts based on your network requirements. With validation, you can update the target configuration buffer with any modifications that are made by the config scripts. You can review the target configuration using the **show configuration** command, and further refine the changes to resolve any outstanding errors before revalidating or committing the configuration.

Note If the config script rejects one or more items in the commit operation, the entire commit operation is rejected.

You can also validate pre-configuration during a commit operation. Pre-configuration is any configuration specific to a particular hardware resource such as an interface or a line card that is committed before that resource is present. For example, commit configuration for a line card before it is inserted into the chassis. Any active config scripts can read and validate (accept, reject or modify) the pre-configuration. However, when the configuration is committed, the pre-configuration is not applied to the system. Later, when the relevant hardware resource is available, the pre-configuration becomes active and is applied to the system. The config scripts are not run to validate the configuration at this point as the scripts have already validated this configuration.

Before you begin

Ensure that the following prerequisites are met before you run the script:

1. Enable Config Scripts Feature, on page 181
2. Configure Checksum for Config Script, on page 184

Step 1 Validate the configuration with the conditions in the config script.

Example:

```
Router(config)#validate config-scripts apply-policy-modifications
Tue Aug 31 08:30:38.613 UTC

% Policy modifications were made to target configuration, please issue 'show configuration'
from this session to view the resulting configuration
                             figuration' from this session to view the resulting configuration
```

The output shows that there are no errors in the changed configuration. You can view the modifications made to the target configuration.

Note If you do not want the config buffer to be updated with the modifications, omit the **apply-policy-modifications** keyword in the command.

The script validates the configuration changes with the conditions set in the script. Based on the configuration, the script stops the commit operation, or modifies the configuration.

Step 2 View the modified target configuration.

Example:

```
Router(config)#show configuration
Tue Aug 31 08:30:56.833 UTC
Building configuration...
!! IOS XR Configuration 7.3.2
script config config-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
                                          d342adb35cbc8a0cd4b6ea1063d0eda2d58
.....----- configuration details
end
```

Step 3 Commit the configuration.

Example:

```
Router(config)#commit
Tue Aug 31 08:31:32.926 UTC
```

If the script returns an error, use the **show configuration failed if-committed** command to view the errors. If there are no validation errors, the commit operation is successful including any modifications that are made by config scripts.

You can view the recent commit operation that the script modified, and display the original configuration changes before the script modified the values using **show configuration commit changes original last-modified** command.

If the commit operation is successful, you can check what changes were committed including the script modifications using **show configuration commit changes last 1** command.

Note If a config script returns a modified value that is syntactically invalid, such as an integer that is out of range, then the configuration is not converted to CLI format for use in operational commands. This action impacts the **validate config-scripts apply-policy-modifications** command and **show configuration** command to view the modifications, and **show configuration failed [if-committed]** command during a failed commit operation.

Step 4 After the configuration change is successful, view the running configuration and logs for details.

Example:

```
Router(config)#show logging
Tue Aug 31 08:31:54.472 UTC
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
```

```
    Console logging: Disabled
    Monitor logging: level debugging, 0 messages logged
    Trap logging: level informational, 0 messages logged
    Buffer logging: level debugging, 13 messages logged

Log Buffer (2097152 bytes):
-------------------- snipped for brevity ----------------------------------
Configuration committed by user 'cisco'. Use 'show configuration commit changes
1000000006' to view the changes.
```

Manage Scripts

This section shows the additional operations that you can perform on a script.

Delete Config Script from the Router

You can delete a config script from the script management repository using the **script remove** command.

Step 1 View the active scripts on the router.

Example:

```
Router#show script status
Wed Aug 24 10:10:50.453 UTC
=============================================================================================================

 Name                            | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 ssh_config_script.py            | config | Ready            | NEW         | Tue Aug 24 09:18:23 2021

 =============================================================================================================
```

Ensure the script that you want to delete is present in the repository.

Alternatively you can also view the list of scripts from the IOS XR Linux bash shell.

```
[node0_RP0_CPU0:/harddisk:/mirror/script-mgmt/config]$ls -lrt
total 1
-rw-rw-rw-. 1 root root 110 Aug 24 10:44 ssh_config_script.py
```

Step 2 Delete script `ssh_config_script.py`.

Example:

```
Router#script remove config ssh_config_script.py
Tue Aug 24 10:19:38.170 UTC
ssh_config_script.py has been deleted from the script repository
```

You can also delete multiple scripts simultaneously.

```
Router#script remove config sample1.py sample2.py sample3.py
```

Step 3 Verify that the script is deleted from the subdirectory.

Example:

```
Router#show script status
Tue Aug 24 10:24:38.170 UTC
### No scripts found ###
```

The script is deleted from the script management repository.

If a config script is still configured when it is removed, subsequent commit operations are rejected. So, you must also undo the configuration of the script:

```
Router(config)#no script config ssh_config_script.py
Router(config)#commit
```

Control Priority When Running Multiple Scripts

If the set of active scripts includes two (or more) that may attempt to modify the same configuration item but to different values, whichever script runs last takes precedence. The script that was last run supersedes the values written by the script (or scripts) that ran before it. It is recommended to avoid such dependencies between scripts. For example, you can combine such scripts into a single script. If the dependency cannot be resolved, you can specify which script takes precedence by ensuring it runs last.

Priority can also be used to ensure scripts run in an optimal order, which may be important if scripts consume resources and impacts performance. For example, consider that script A sets configuration that is validated by script B. Without a set priority, the system may run script B first, then script A, and then script B a second time to validate the changes made by script A. With a configured priority, the system ensures that script A runs first, and script B needs to run only once.

The priority value is an integer between 0-4294967295. The default value is 500.

Consider script `sample1.py` depends on `sample2.py` to validate the configuration that the script sets. The script `sample1.py` must be run first, followed by `sample2.py`. Configure the priority to ensure that the system runs the scripts in a specified order.

Step 1 Configure script `sample1.py` with a lower priority.

Example:

```
Router(config)#script config sample1.py checksum sha256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
priority 10
```

Step 2 Configure script `sample2.py` with a higher priority.

Example:

```
Router(config)#script config sample2.py checksum sha256
2fa34b64542f005ed58dcaa1f3560e92a03855223e130535978f8c35bc21290c
priority 20
```

Step 3 Commit the configuration.

Example:

```
Router(config)#commit
```

The system checks the priority values, and runs the one with lower priority first (`sample1.py`), followed by the one with the higher priority value (`sample2.py`).

Example: Validate and Activate an SSH Config Script

This section presents examples for config script that enforces various constraints related to SSH configuration, including making modifications to the configuration in some cases. The following sub-sections illustrate the behaviour of this script in various scenarios.

Before you begin

Ensure you have completed the following prerequisites before you validate the script:

1. Enable config scripts feature on the router. See Enable Config Scripts Feature, on page 181.
2. Create a config script `ssh_config_script.py`. Store the script on an HTTP server or copy the script to the harddisk of the router.

```
import cisco.config_validation as xr
from cisco.script_mgmt import xrlog
syslog = xrlog.getSysLogger('xr_cli_config')

def check_ssh_late_cb(root):
    SSH = "/crypto-ssh-cfg:ssh"
    SERVER = "/crypto-ssh-cfg:ssh/server"
    SESSION_LIMIT = "session-limit"
    LOGGING = "logging"
    RATE_LIMIT = "rate-limit"
    V2 = "v2"
    server = root.get_node(SERVER)
    if server is None:
        xr.add_error(SSH, "SSH must be enabled.")

    if server :
        session_limit = server.get_node(SESSION_LIMIT)
        rate_limit = server.get_node(RATE_LIMIT)
        ssh_logging = server.get_node(LOGGING)
        ssh_v2 = server.get_node(V2)

        if session_limit is None or session_limit.value >= 100:
            server.set_node(SESSION_LIMIT, 80)
        if rate_limit.value == 60:
            xr.add_warning(rate_limit, "RATE_LIMIT should not be set to default value")

        if not ssh_logging:
            server.set_node(LOGGING)
        if not ssh_v2:
            xr.add_error(server, "Server V2 need to be set")

xr.register_validate_callback(["/crypto-ssh-cfg:ssh/server/*"], check_ssh_late_cb)
```

The script checks the following actions:

- Check if SSH is enabled. If not, generate an error message `SSH must be enabled` and stop the commit operation.

- Check if the rate-limit is set to 60, display a warning message that the `RATE_LIMIT should not be set to default value` and allow the commit operation.
- Check if the session-limit is set. If the limit is 100 sessions or more, set the value to 80 and allow the commit operation.
- Set the logging if not already enabled.

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

Scenario 1: Validate the Script Without SSH Configuration

In this example, you validate a script without SSH configuration. The script is programmed to check the SSH configuration. If not configured, the script instructs the system to display an error message and stop the commit operation until SSH is configured.

Step 1 Configure the checksum to verify the authenticity and integrity of the script. See Configure Checksum for Config Script, on page 184.

Step 2 Validate the config script.

Example:

```
Router(config)#validate config-scripts apply-policy-modifications
Wed Sep 1 23:21:34.730 UTC

% Validation of configuration items failed. Please issue 'show configuration failed if-committed'
from this
session to view the errors
```

The validation of the configuration failed.

Step 3 View the configuration of the failed operation.

Example:

```
Router#show configuration failed if-committed
Wed Sep 1 22:01:07.492 UTC
!! SEMANTIC ERRORS: This configuration was rejected by !! the system due to semantic errors.
!! The individual errors with each failed configuration command can be found below.

script config ssh_config_script.py checksum SHA256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
!!% ERROR: SSH must be enabled.
end
```

The message for the failure is displayed. Here, the error `SSH must be enabled` is displayed as programmed in the script. The script stops the commit operation because the changes do not comply with the rule set in the script.

Step 4 Check the syslog output for the count of errors, warnings, and modifications.

Example:

```
Router#show logging  | in Error
Wed Sep 1 22:02:05.559 UTC
Router:Wed Sep 1 22:45:05.559 UTC: ccv[394]: %MGBL-CCV-6-CONFIG_SCRIPT_CALLBACK_EXECUTED :
The function check_ssh_late_cb registered by the config script ssh_config_script.py was
executed in 0.000 seconds.
Error/Warning/Modification counts: 1/0/0
```

In this example, the script displays an error about the missing SSH configuration. When an error is displayed, the warning and modification count always show 0/0 respectively even if modifications exist on the target buffer.

Scenario 2: Configure SSH and Validate the Script

In this example, you configure SSH to resolve the error displayed in scenario 1, and validate the script again.

Step 1 Configure SSH.

Example:

```
Router(config)#ssh server v2
Router(config)#ssh server vrf default
Router(config)#ssh server netconf vrf default
```

Step 2 Configure the checksum.

Step 3 Validate the configuration again.

Example:

```
Router(config)#validate config-scripts apply-policy-modifications
Wed Sep 1 22:03:05.448 UTC

% Policy modifications were made to target configuration, please issue 'show configuration'
from this session to view the resulting configuration
```

The script is programmed to display an error and stop the commit operation if the system detects that SSH server is not configured. After the SSH server is configured, the script is validated successfully.

Step 4 Commit the configuration.

Example:

```
Router(config)#commit
Tue Aug 31 08:31:32.926 UTC
```

Step 5 View the SSH configuration that is applied or modified after the commit operation.

Example:

```
Router#show running-config ssh
Wed Sep 1 22:15:05.448 UTC
ssh server logging
ssh server session-limit 80
ssh server v2
ssh server vrf default
ssh server netconf vrf default
```

In addition, you see the modifications that are made by the script to the target buffer. The session-limit is used to configure the number of allowable concurrent incoming SSH sessions. In this example, the default limit is set to 80 sessions. Outgoing connections are not part of the limit. The script is programmed to check the session limit. If the limit is greater or equal to 100 sessions, the script reconfigures the value to the default 80 sessions. However, if the limit is within 100 sessions, the configuration is accepted without modification.

Step 6 Check the syslog output for the count of errors, warnings, and modifications.

Example:

```
Router#show logging  | in Error
Wed Sep 1 22:45:05.559 UTC
```

```
Router:Wed Sep 1 22:45:05.559 UTC: ccv[394]: %MGBL-CCV-6-CONFIG_SCRIPT_CALLBACK_EXECUTED :
The function check_ssh_late_cb registered by the config script ssh_config_script.py was
executed in 0.000 seconds.
Error/Warning/Modification counts: 0/0/2
```

In this example, the script did not display an error or warning, but made two modifications for server logging and session-limit.

Scenario 3: Set Rate-limit Value to Default Value in the Script

In this example, you see the response after setting the rate-limit to the default value configured in the script. The rate-limit is used to limit the incoming SSH connection requests to the configured rate. The SSH server rejects any connection request beyond the rate-limit. Changing the rate-limit does not affect established SSH sessions. For example, if the rate-limit argument is set to 60, then 60 requests are allowed per minute. The script checks if the rate-limit is set to the default value 60. If yes, the script displays a warning message that the `RATE_LIMIT should not be set to default value`, but allow the commit operation.

Step 1 Configure rate-limit to the default value of 60.

Example:

```
Router(config)#ssh server rate-limit 60
```

Step 2 Commit the configuration.

Example:

```
Router(config)#commit
Wed Sep 1 22:11:05.448 UTC

% Validation warnings detected as a result of the commit operation.
Please issue 'show configuration warnings' to view the warnings
```

The script displays a warning message but proceeds with the commit operation.

Step 3 View the warning message.

Example:

```
Router(config)#show configuration warnings
Wed Sep 1 22:12:05.448 UTC
!! SEMANTIC ERRORS: This configuration was rejected by the system due to
semantic errors. The individual errors with each failed configuration command
can be found below.

script config ssh_config_script.py checksum SHA256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
!!% WARNING: RATE_LIMIT should not be set to default value
end
```

The rate limit is default value of 60. The script is programmed to display a warning message if the rate limit is set to the default value. You can either change the limit or leave the value as is.

Step 4 View the running configuration.

Example:

```
Router(config)#do show running-config  script
Wed Sep 1 22:15:05.448 UTC
script config ssh_config_script.py checksum SHA256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
```

The script `ssh_config_script.py` is active.

Scenario 4: Delete SSH Server Configuration

In this example, you delete the SSH server configurations, and see the response when the script is validated.

Step 1 Remove the SSH server configuration.

Example:

```
Router(config)#no ssh server v2
```

Step 2 Commit the configuration.

Example:

```
Router(config)#commit
Wed Sep 1 22:45:05.559 UTC

% Failed to commit one or more configuration items during an atomic operation.
No changes have been made. Please issue 'show configuration failed if-committed' from
this session to view the errors
```

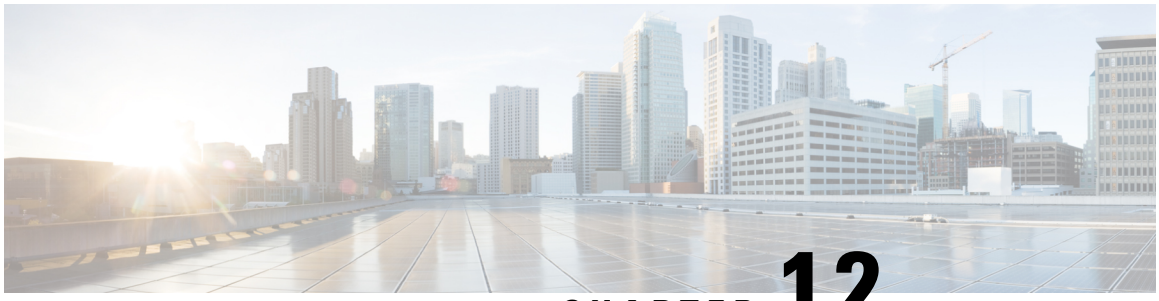
Step 3 View the error message.

Example:

```
Router(config)#show configuration failed if-committed
Wed Sep 1 22:47:53.202 UTC
!! SEMANTIC ERRORS: This configuration was rejected by the system due to semantic errors. The individual
errors with each failed configuration command can be found below.

no ssh server v2
!!% ERROR: Server V2 need to be set
end
```

The message is displayed based on the rule set in the script.



CHAPTER 12

Exec Scripts

Cisco IOS XR exec scripts are on-box scripts that automate configurations of devices in the network. The exec scripts are written in Python using the Python libraries that Cisco provides with the base package. For the list of supported packages

A script management repository on the router manages the exec scripts. This repository is replicated on both RPs.

In IOS XR, AAA authorization controls the user access and privileges to perform operations. To run the exec script, you must have root user permissions.

Exec scripts provide the following advantages:

- Provides automation capabilities to simplify complex operations.
- Create customized operations based on the requirement.
- Provide flexibility in changing the input parameters for every script run. This fosters dynamic automation of operational information.
- Detect and display errors and warnings when executing an operation.
- Run multiple automated operations in parallel without blocking the console.

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

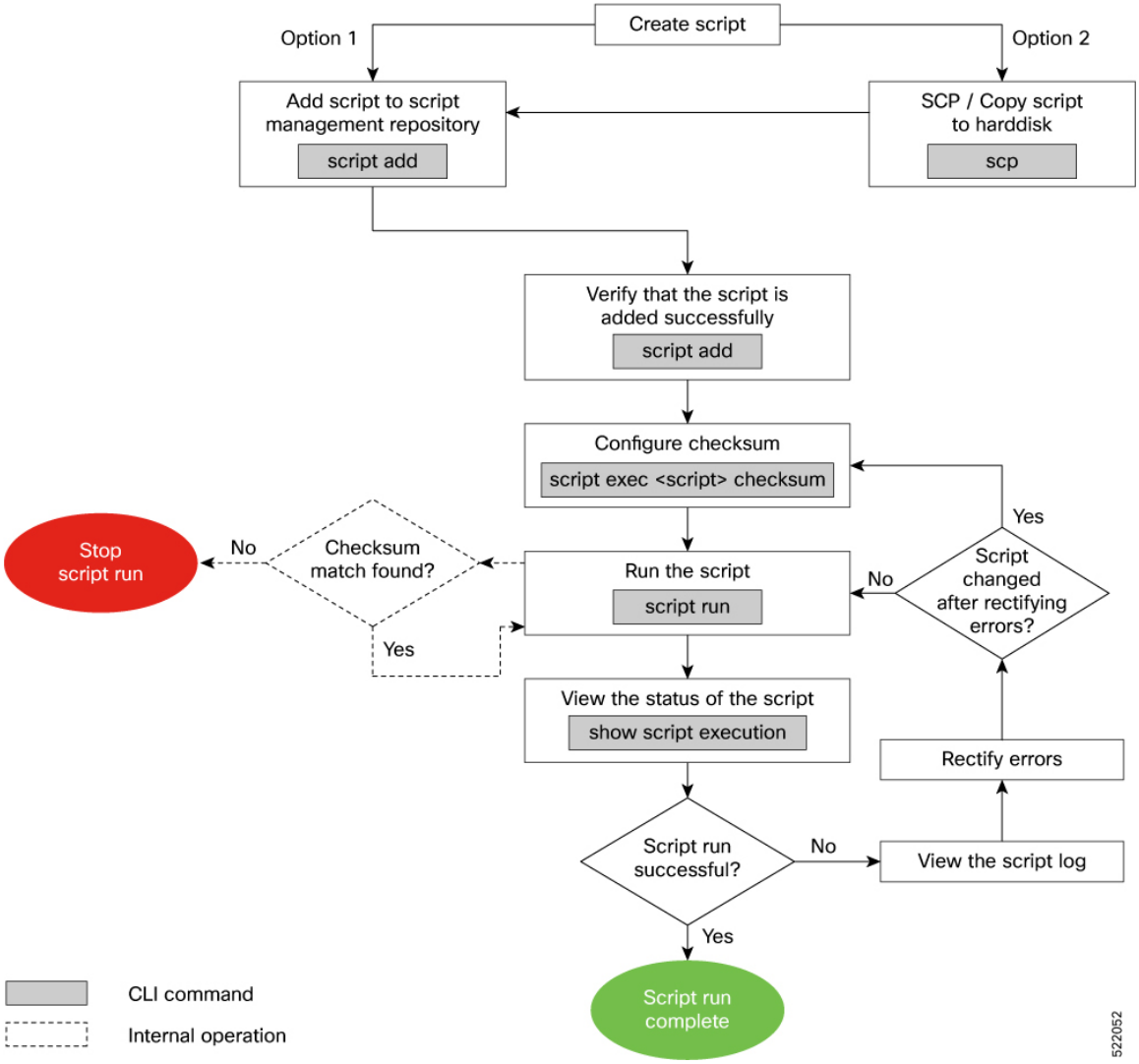
Note This chapter does not delve into creating Python scripts, but assumes that you have basic understanding of Python programming language. This section will walk you through the process involved in deploying and using the scripts on the router.

Workflow to Run an Exec Script

Complete the following tasks to provision exec scripts:

- Download the script—Add the script to the appropriate exec script directory on the router. using the **script add exec** command.
- Configure checksum—Check script integrity and authenticity using the **script exec <script.py> checksum** command.
- Run the script—Trigger changes to the router configuration. Include arguments, set the maximum time for the script to run, setup log levels using the **script run** command.
- View the script execution details—Validate the script and retrieve the operational data using the **show script execution** command.

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



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 `exec-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 exec <script-location> <script.py>
  ```

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

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

  Note The repository can be local to the router, or accessed remotely through TFTP, SCP, FTP, HTTP, or HTTPS protocols. In addition to the default Virtual Routing and Forwarding (VRF), support is also extended for non-default VRF.

  You can add a maximum of 10 scripts simultaneously.

  ```
  Router#script add exec <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.

  Note Only SHA256 checksum is supported.

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

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

```
Router#script add exec 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.

- **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/exec-script.py /harddisk:/
  ```

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

  ```
  Router#script add exec /harddisk:/ exec-script.py
  Fri Aug 20 05:03:40.791 UTC
  exec-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
---------------------------------------------------------------------------------------------
 exec-script.py    | exec     | Config Checksum  | NEW         | Tue Aug 24 10:18:23 2021
 =============================================================================================
```

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

Update Scripts from a Remote Server

Table 50: Feature History Table

Feature Name	Release Information	Description
Update Automation Scripts from Remote Server	Release 7.5.1	This feature lets you update automation scripts across routers by accessing the master script from a remote site. This eases script management, where you make changes to the master script and then copy it to routers where it is deployed. This feature introduces the **auto-update** keyword in the **script exec** command.

You can maintain the latest copy of the scripts in a remote location, and configure the routers to automatically update the local copy with the latest copy on the server as required.

You can update the script using one of the following options.

- **Config CLI commands:**

  a. Update the script on the router with the version on the remote server.

  ```
  Router(config)#script exec auto-update sample3.py http://10.23.255.205
  condition [manual | on-run | schedule]
  ```

  In this example, `sample3.py` script is automatically updated from the remote server at `http://10.23.255.205`. You can set conditions when updating the script.

  The repository can be accessed remotely through FTP, HTTP, HTTPS, TFTP or SCP protocols.

Condition	Description
manual	Update manually with an Exec CLI (default). The following option is supported: • vrf—Specify the non-default Virtual Routing and Forwarding (VRF) name. • username—Enter the username. • password—Enter the password.
on-run	Update the exec script during run time. The following options are supported: • on-fail—Specify one of the actions on failure. • do-not-run—Do not run the script on failure. • run-local—Run the local copy of the script. • vrf—Specify the non-default VRF name. • username—Enter the username. • password—Enter the password. **Note** Only the exec scripts support the **on-run** option.
schedule	Update automatically at specified time intervals. The following option is supported: • <60-262800>—Update interval in minutes • username—Enter the username. • password—Enter the password. **Note** The **schedule** option does not support SCP protocol.

Note Do not specify the username and password inside the URL of the remote server.

b. Commit the configuration.

```
Router(config)#commit
```

c. Run the script.

```
Router#script run sample3.py background
Tue Nov 16 12:50:33.512 UTC
sample3.py has been added to the script repository
Script run scheduled: sample3.py. Request ID: 1624990452
```

You can specify additional options to the command:

- **arguments:** Script command-line arguments. The format is strings in single quotes. Escape double quotes inside string arguments.
- **description:** Description of script run.
- **log-level:** Script logging level. Default is INFO.
- **log-path:** Location to store script logs.
- **max-runtime:** Maximum run time of script.

- **Exec CLI commands:**

When you run the script, the script is downloaded and the checksum is automatically configured on the router.

- If **on-run** option is configured, running the **script run** command downloads the script.
- If **manual** option is configured, then you must run **script update** Exec command.
- If **schedule** option is selected, then the script is automatically updated after the specified interval.

a. Update the script on the router with the version on the remote server.

```
Router#script update manual exec sample2.py
Tue Nov 16 12:20:23.058 UTC
sample2.py has been added to the script repository
```

You can set options when updating the script:

Option	Description
WORD	Script name.
all	Update all scripts in config.

Invoke Scripts from a Remote Server

You can directly run the script using the URL to the remote server and provide the checksum value. The checksum is a mandatory parameter. The format of the URL is `[protocol]://[user:password@]server[:port]/directory/file_name`.

Run the script from the remote server.

Example:

```
Router#script run http://10.23.255.205/sample1.py checksum
5103a843032505decc37ff21089336e4bcc6a1061341056ca8add3ac5d6620ef background
Tue Nov 16 12:12:08.614 UTC
Script run scheduled: sample1.py. Request ID: 1624990451
```

The repository can be accessed remotely through FTP, HTTP, HTTPS, TFTP or SCP protocols.

You can specify additional options to the command:

- **arguments:** Script command-line arguments. The format is strings in single quotes. Escape double quotes inside string arguments.
- **description:** Description of script run.
- **log-level:** Script logging level. Default is INFO.
- **log-path:** Location to store script logs.
- **max-runtime:** Maximum run time of script.
- **vrf:** Specify the VRF for the network file system.

Configure Checksum for Exec Script

Every script is associated with a checksum value. The checksum ensures the integrity of the script that is downloaded from the server or external repository is intact, and that the script is not tampered. The checksum is a string of numbers and letters that act as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a syslog warning message is displayed.

It is mandatory to configure the checksum to run the script.

Note Exec scripts support SHA256 checksum.

Before you begin

Ensure that the script is added to the script management repository. See Download the Script to the Router, on page 197.

Step 1 Retrieve the SHA256 checksum hash value for the script. Ideally this action would be performed on a trusted device, such as the system on which the script was created. This minimizes the possibility that the script is tampered with.

Example:

```
Server$sha256sum sample1.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b sample1.py
```

Make note of the checksum value.

Step 2 View the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:04:13.539 UTC
=============================================================================================================

 Name                            | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 sample1.py                      | exec   | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : sample1.py
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The `Status` shows that the checksum is not configured.

Step 3 Enter global configuration mode.

Example:

```
Router#configure
```

Step 4 Configure the checksum.

Example:

```
Router(config)#script exec sample1.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Tue Aug 24 10:23:10.546 UTC
Router(config)#end
```

Step 5 Verify the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:06:17.296 UTC
=============================================================================================================

 Name                            | Type   | Status           | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------

 sample1.py                      | exec   | Ready            | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : cpu_load.py
 Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Checksum     : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script fails. It is mandatory for the checksum values to match for the script to run.

Run the Exec Script

To run an exec script, use the **script run** command. After the script is run, a request ID is generated. Each script run is associated with a unique request ID.

Before you begin

Ensure the following prerequisites are met before you run the script:

1. Download the Script to the Router, on page 197
2. Configure Checksum for Exec Script, on page 201

Run the exec script.

Example:

```
Router#script run sample1.py
Wed Aug 25 16:40:59.134 UTC
Script run scheduled: sample1.py. Request ID: 1629800603
Script sample1.py (exec) Execution complete: (Req. ID 1629800603) : Return Value: 0 (Executed)
```

Scripts can be run with more options. The following table lists the various options that you can provide at run time:

Keyword	Description
arguments	Script command-line arguments. Syntax: Strings in single quotes. Escape double quotes inside string arguments (if any). For example: `Router#script run sample1.py arguments 'hello world' '-r' '-t' 'exec' '--sleep' '5' description "Sample exec script"`
background	Run script in background. By default, the script runs in the foreground. When a script is run in the background, the console is accessible only after the script run is complete.
description	Description about the script run. `Router#script run sample1.py arguments '-arg1' 'reload' '-arg2' 'all' 'description' "Script reloads the router"` When you provide both the argument and description ensure that the arguments are in single quote and description is in double quotes.

Keyword	Description
log-level	Script logging level. The default value is `INFO`. You can specifiy what information is to be logged. The log level can be set to one of these options—Critical, Debug, Error, Info, or Warning.
log-path	Location to store the script logs. The default log file location is in the script management repository `harddisk:/mirror/script-mgmt/logs`.
max-runtime	Maximum run-time of script can be set between 1–3600 seconds. The default value is 300.

The script run is complete.

View the Script Execution Details

View the status of the script execution.

Before you begin

Ensure the following prerequisites are met before you run the script:

1. Download the Script to the Router, on page 197
2. Configure Checksum for Exec Script, on page 201
3. Run the Exec Script, on page 203

Step 1 View the status of the script execution.

Example:

```
Router#show script execution
Wed Aug 25 18:32:12.351 UTC
==========================================================================================================

 Req. ID    | Name (type)              | Start                    | Duration    | Return | Status
----------------------------------------------------------------------------------------------------------

 1629800603| sample1.py (exec)         | Wed Aug 25 16:40:59 2021 | 60.62s      | 0      | Executed
==========================================================================================================
```

You can view detailed or filtered data for every script run.

Step 2 Filter the script execution status to view the detailed output of a specific script run via request ID.

Example:

```
Router#show script execution request-id 1629800603 detail output
Wed Aug 25 18:37:12.920 UTC
==========================================================================================================

 Req. ID    | Name (type)                              | Start                    | Duration    |
Return | Status
----------------------------------------------------------------------------------------------------------
```

```
1629800603| sample1.py (exec)                          | Wed Aug 25 16:40:59 2021 | 60.62s      | 0
    | Executed
-----------------------------------------------------------------------------------------------------------------

Execution Details:
------------------
Script Name  : sample1.py
Log location : /harddisk:/mirror/script-mgmt/logs/sample1.py_exec_1629800603
Arguments    :
Run Options  : Logging level - INFO, Max. Runtime - 300s, Mode - Foreground
Events:
-------
1.   Event        : New
     Time         : Wed Aug 25 16:40:59 2021
     Time Elapsed : 0.00s Seconds
     Description  : None
2.   Event        : Started
     Time         : Wed Aug 25 16:40:59 2021
     Time Elapsed : 0.03s Seconds
     Description  : Script execution started. PID (20736)
3.   Event        : Executed
     Time         : Wed Aug 25 16:42:00 2021
     Time Elapsed : 60.62s Seconds
     Description  : Script execution complete
-----------------------------------------------------------------------------------------------------------------

Script Output:
--------------
Output File  : /harddisk:/mirror/script-mgmt/logs/sample1.py_exec_1629800603/stdout.log
Content      :
=================================================================================================================
```

Keyword	Description
detail	Display detailed script execution history, errors, output and deleted scripts. `Router#`**`show script execution detail [errors | output | show-del]`**
last <number>	Show last N (1-100) execution requests. `Router#`**`show script execution last 10`** This example will display the list of last 10 script runs with their request IDs, type of script, timestamp, duration that the script was run, number of errrors, and the status of the script run.
name <filename>	Filter operational data based on script name. If not specified, all scripts are displayed. `Router#`**`show script execution name sample1.py`**
request-id <value>	Display summary of the script using request-ID that is generated with each script run. `Router#`**`show script execution request-ID 1629800603`**
reverse	Display the request IDs from the script execution in reverse chronological order. For example, the request-ID from the latest run is displayed first, followed by the descending order of request-IDs. `Router#`**`script script execution reverse`**

Keyword	Description
status	Filter data based on script status. `Router#[status {Exception, Executed, Killed, Started, Stopped, Timed-out}]`

Manage Scripts

This section shows the additional operations that you can perform on a script.

Delete Exec Script from the Router

Delete the script from the script management repository using the **script remove** command. This repository stores the downloaded scripts.

Step 1 View the list of scripts present in the script management repository.

Example:

```
Router#show script status
Wed Aug 25 23:10:50.453 UTC
=============================================================================================================
Name       | Type   | Status          | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------
sample1.py | exec   | Config Checksum | NEW         | Tue Aug 24 10:18:23 2021
sample2.py | exec   | Config Checksum | NEW         | Wed Aug 25 23:44:53 2021
sample3.py | config | Config Checksum | NEW         | Wed Aug 25 23:44:57 2021
```

Ensure the script you want to delete is present in the repository.

Step 2 Delete the script.

Example:

```
Router#script remove exec sample2.py
Wed Aug 25 231:46:38.170 UTC
sample2.py has been deleted from the script repository
```

You can also delete multiple scripts simulataneoulsy.

Step 3 Verify the script is deleted from the subdirectory.

Example:

```
Router#show script status
Wed Aug 25 23:48:50.453 UTC
=============================================================================================================
Name       | Type   | Status          | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------
sample1.py | exec   | Config Checksum | NEW         | Tue Aug 24 10:18:23 2021
sample3.py | config | Config Checksum | NEW         | Wed Aug 25 10:44:57 2021
```

The script is deleted from the script management repository.

Example: Exec Script to Verify Bundle Interfaces

In this example, you create a script to verify the bandwidth usage of bundle interfaces on the router, and check if it is beyond the defined limit. If usage is above the limit, the script generates a syslog indicating that the bandwidth is above the limit, and additional interfaces must be added to the bundle.

Before you begin

Ensure you have completed the following prerequisites before you validate the script:

1. Create an exec script `verify_bundle.py`. Store the script on an HTTP server or copy the script to the harddisk of the router.

```
"""
Bundle interfaces bandwidth verification script

Verify bundle interfaces mpls packets per sec is below threshold.
If pkts/sec is greater than threshold then print syslog message
and add list of new interfaces to bundle

Arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  Bundle interface name
  -t THRESHOLD, --threshold THRESHOLD
                        Bandwidth threshold
  -m MEMBERS, --members MEMBERS
                        interfaces (coma separated) to add to bundle
"""
import re
import argparse
from iosxr.xrcli.xrcli_helper import XrcliHelper
from cisco.script_mgmt import xrlog

syslog = xrlog.getSysLogger('verify_bundle')
log = xrlog.getScriptLogger('verify_bundle')


def add_bundle_members(bundle_name, members):

    helper = XrcliHelper()
    bundle_pattern = re.compile('[A-Z,a-z, ]([0-9]+)')
    match = bundle_pattern.search(bundle_name)
    if match:
        bundle_id = match.group(1)
    else:
        raise Exception('Invalid bundle name')
    cfg = ''
    for member in members:

        cfg = cfg + 'interface %s \nbundle id %s mode active\nno shutdown\n' % \
              (member.strip(), bundle_id)

    log.info("Configs to be added : \n%s" % cfg)
    result = helper.xr_apply_config_string(cfg)
    if result['status'] == 'success':
        msg = "Configuring new bundle members successful"
        syslog.info(msg)
        log.info(msg)
    else:
        msg = "Configuring new bundle members failed"
        syslog.warning(msg)
```

```
            log.warning(msg)


    def verify_bundle(script_args):

        helper = XrcliHelper()
        cmd = "show interfaces %s accounting rates" % script_args.name
        cmd_out = helper.xrcli_exec(cmd)
        if not cmd_out['status'] == 'success':
            raise Exception('Invalid bundle or error getting interface accounting rates')

        log.info('Command output : \n%s' % cmd_out['output'])
        rate_pattern = re.compile("MPLS +[0-9]+ +[0-9]+ +[0-9]+ +([0-9]+)")
        match = rate_pattern.search(cmd_out['output'])
        if match:
            pktspersec = int(match.group(1))
            if pktspersec > int(script_args.threshold):
                msg = 'Bundle %s bandwidth of %d pps is above threshold of %s pps' % \
                        (script_args.name, pktspersec, script_args.threshold)
                log.info(msg)
                syslog.info(msg)
                return False
            else:
                msg = 'Bundle %s bandwidth of %d pps is below threshold of %s pps' % \
                        (script_args.name, pktspersec, script_args.threshold)
                log.info(msg)
                return True


    if __name__ == '__main__':

        parser = argparse.ArgumentParser(description="Verify budle")
        parser.add_argument("-n", "--name",
                            help="Bundle interface name")
        parser.add_argument("-t", "--threshold",
                            help="Bandwidth threshold")
        parser.add_argument("-m", "--members",
                            help="interfaces (coma separated) to add to bundle")
        args = parser.parse_args()
        log.info('Script arguments :')
        log.info(args)
        if not verify_bundle(args):
            syslog.info("Adding new members (%s) to bundle interfaces %s" %
                        (args.members, args.name))
            add_bundle_members(args.name, args.members.split(','))
```

2. Add the script from HTTP server or harddisk to the script management repository. See Download the Script to the Router, on page 197.
3. Configure the checksum to verify the authenticity and integrity of the script.

Step 1 View the script status.

Example:

```
Router#show script status
Sat Sep 25 00:10:11.222 UTC
================================================================================
Name                | Type   | Status   | Last Action | Action Time
--------------------------------------------------------------------------------
verify_bundle.py    | exec   | Ready    | MODIFY      | Sat Sep 25 00:08:55 2021
================================================================================
```

The status indicates that the script is ready to be run.

Step 2 Run the script.

Example:

```
Router#script run verify_bundle.py arguments '--name' 'Bundle-Ether6432' '-t'
'400000' '-m' 'FourHundredGigE0/0/0/2
Sat Sep 25 00:11:14.183 UTC
Script run scheduled: verify_bundle.py. Request ID: 1632528674
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Script arguments :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Namespace(members='FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3', name='Bundle-Ether6432', threshold='400000')
[2021-09-25 00:11:14,735] INFO [verify_bundle]:: Command output :

-------------- show interfaces Bundle-Ether6432 accounting rates --------------
Bundle-Ether6432
                              Ingress                        Egress
  Protocol             Bits/sec         Pkts/sec     Bits/sec         Pkts/sec
  IPV4_UNICAST            22000               40            0                0
  MPLS                        0                0   1979249000           430742
  ARP                         0                0            0                0
  IPV6_ND                     0                0            0                0
  CLNS                     1000                1        26000                3



[2021-09-25 00:11:14,736] INFO [verify_bundle]:: Bundle Bundle-Ether6432 bandwidth
of 430742 pps is above threshold of 400000 pps
[2021-09-25 00:11:14,737] INFO [verify_bundle]:: Configs to be added :
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
no shutdown
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
no shutdown

[2021-09-25 00:11:18,254] INFO [verify_bundle]:: Configuring new bundle members successful
Script verify_bundle.py (exec) Execution complete: (Req. ID 1632528674) : Return Value: 0 (Executed)
```

Step 3 View the detailed output based on request ID. A request ID is generated for each script run.

Example:

```
Router#show script execution request-id 1632528674 detail output
Sat Sep 25 00:11:58.141 UTC
================================================================================================
Req. ID   | Name (type)               | Start                    | Duration   | Return | Status
------------------------------------------------------------------------------------------------
1632528674| verify_bundle.py (exec)    | Sat Sep 25 00:11:14 2021 | 4.06s      | 0      | Executed
------------------------------------------------------------------------------------------------
Execution Details:
------------------
Script Name  : verify_bundle.py
Log location : /harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674
Arguments    : '--name', 'Bundle-Ether6432', '-t', '400000', '-m', 'FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3'
Run Options  : Logging level - INFO, Max. Runtime - 300s, Mode - Foreground
Events:
-------
1.   Event        : New
      Time         : Sat Sep 25 00:11:14 2021
      Time Elapsed : 0.00s Seconds
      Description  : None
```

```
2.   Event        : Started
      Time         : Sat Sep 25 00:11:14 2021
      Time Elapsed : 0.02s Seconds
      Description  : Script execution started. PID (29768)
3.   Event        : Executed
      Time         : Sat Sep 25 00:11:18 2021
      Time Elapsed : 4.06s Seconds
      Description  : Script execution complete
------------------------------------------------------------------------------------------------------------------------
Script Output:
--------------
Output File  : /harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674/stdout.log
Content      :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Script arguments :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Namespace(members='FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3',
name='Bundle-Ether6432', threshold='400000')
[2021-09-25 00:11:14,735] INFO [verify_bundle]:: Command output :

-------------- show interfaces Bundle-Ether6432 accounting rates --------------
Bundle-Ether6432
                               Ingress                        Egress
  Protocol             Bits/sec         Pkts/sec     Bits/sec         Pkts/sec
  IPV4_UNICAST            22000               40            0                0
  MPLS                        0                0   1979249000           430742
  ARP                         0                0            0                0
  IPV6_ND                     0                0            0                0
  CLNS                     1000                1        26000                3

[2021-09-25 00:11:14,736] INFO [verify_bundle]:: Bundle Bundle-Ether6432 bandwidth of 430742 pps is
 above threshold
of 400000 pps
[2021-09-25 00:11:14,737] INFO [verify_bundle]:: Configs to be added :
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
no shutdown
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
no shutdown

[2021-09-25 00:11:18,254] INFO [verify_bundle]:: Configuring new bundle members successful
========================================================================================================================
```

Step 4 View the running configuration for the bundle interfaces.

Example:

```
Router#show running-config interface FourHundredGigE0/0/0/2
Sat Sep 25 00:12:30.765 UTC
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
!

Router#show running-config interface FourHundredGigE0/0/0/3
Sat Sep 25 00:12:38.659 UTC
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
!
```

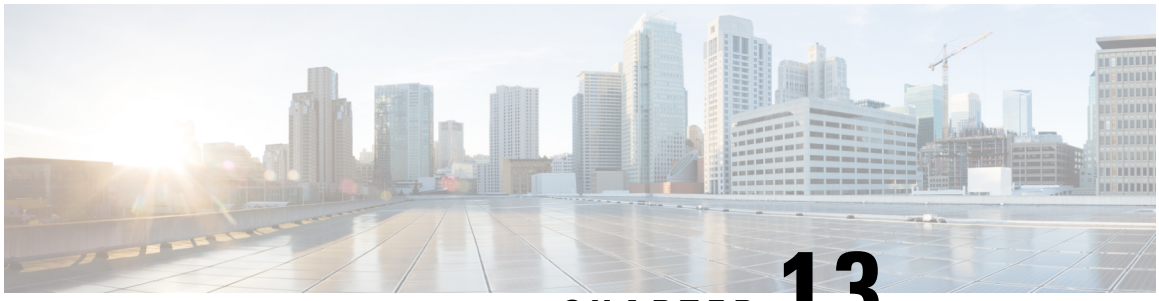
Step 5 View the latest logs for more details about the script run. Here, the last 10 logs are displayed. The logs show that configuring new bundle members is successful.

Example:

```
Router#show logging last 10
Sat Sep 25 00:13:34.383 UTC
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: level warnings, 178 messages logged
    Monitor logging: level debugging, 0 messages logged
    Trap logging: level informational, 0 messages logged
    Buffer logging: level debugging, 801 messages logged

Log Buffer (2097152 bytes):

RP/0/RP0/CPU0:Sep 25 00:10:05.763 UTC: config[66385]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'cisco'.
Use 'show configuration commit changes 1000000045' to view the changes.
RP/0/RP0/CPU0:Sep 25 00:10:07.971 UTC: config[66385]: %MGBL-SYS-5-CONFIG_I : Configured from console
 by cisco on vty0 (6.3.65.175)
RP/0/RP0/CPU0:Sep 25 00:11:14.447 UTC: script_control_cli[66627]: %OS-SCRIPT_MGMT-6-INFO :
Script-control: Script run scheduled:
verify_bundle.py. Request ID: 1632528674
RP/0/RP0/CPU0:Sep 25 00:11:14.453 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Script execution
verify_bundle.py (exec) Started : Request ID : 1632528674 :: PID: 29768
RP/0/RP0/CPU0:Sep 25 00:11:14.453 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Starting execution
verify_bundle.py (exec) (Req. ID: 1632528674) : Logs directory:
/harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674
RP/0/RP0/CPU0:Sep 25 00:11:14.736 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
 Bundle Bundle-Ether6432
bandwidth of 430742 pps is above threshold of 400000 pps
RP/0/RP0/CPU0:Sep 25 00:11:14.736 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
 Adding new members
(FourHundredGigE0/0/0/2, FourHundredGigE0/0/0/3) to bundle interfaces Bundle-Ether6432
RP/0/RP0/CPU0:Sep 25 00:11:16.916 UTC: config[66655]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'cisco'. Use 'show
configuration commit changes 1000000046' to view the changes.
RP/0/RP0/CPU0:Sep 25 00:11:18.254 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
 Configuring new bundle members
successful
RP/0/RP0/CPU0:Sep 25 00:11:18.497 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Script verify_bundle.py
(exec) Execution complete: (Req. ID 1632528674) : Return Value: 0 (Executed)
```



CHAPTER 13

Process Scripts

Cisco IOS XR process scripts are also called daemon scripts. The process scripts are persistent scripts that continue to run as long as you have activated the scripts. An IOS XR process, Application manager (AppMgr or app manager), manages the lifecycle of process scripts. The scripts are registered as an application on the app manager. This application represents the instance of the script that is running on the router.

The app manager is used to:

- Start, stop, monitor, or retrieve the operational status of the script.
- Maintain the startup dependencies between the processes.
- Restart the process if the script terminates unexpectedly based on the configured restart policy.

Process scripts support Python 3.5 programming language. For the list of supported packages, see Cisco IOS XR Python Packages, on page 266.

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

Note This chapter 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. A process script refers to code that runs continuously or endlessly.

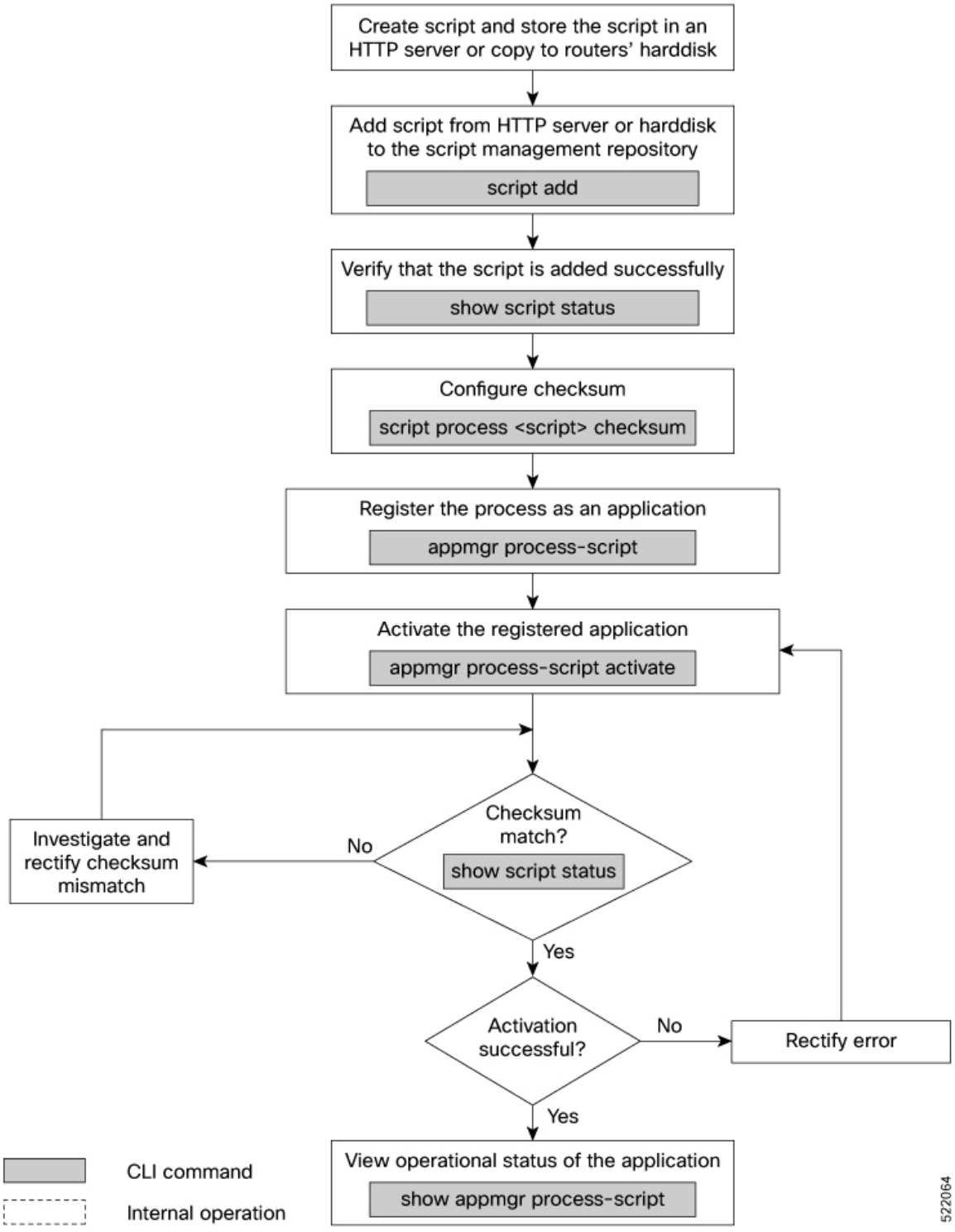
Workflow to Run Process Scripts

Complete the following tasks to provision process scripts:

- Download the script—Store the script on an external server or copy to the harddisk of the router. Add the script from the external server or harddisk to the script management repository on the router using the **script add process** command.
- Configure the checksum—Check script integrity and authenticity using the **script process <script.py> checksum** command.
- Register the script—Register the script as an application in the app manager using **appmgr process-script** command.

- Activate the script—Activate the registered application using **appmgr process-script activate** command.
- View the script execution details—Retrieve the operational data using the **show appmgr process-script** command.

The following image shows the workflow diagram representing the steps that are involved in using a process script:


Create script and store the script in an HTTP server or copy to routers' harddisk
Add script from HTTP server or harddisk to the script management repository
script add
Verify that the script is added successfully
show script status
Configure checksum
script process <script> checksum
Register the process as an application
appmgr process-script
Activate the registered application
appmgr process-script activate
Checksum match?
show script status
No
Investigate and rectify checksum mismatch
Yes
Activation successful?
No
Rectify error
Yes
View operational status of the application
show appmgr process-script
CLI command
Internal operation
522064

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 `process-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 any server or the harddisk location in the router.

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

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

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

  The `script add process` supports the HTTP, HTTPS, FTP, TFTP, and SCP protocols for copying a script.

  You can add a maximum of 10 scripts simultaneously.

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

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

  ```
  Router#script add process 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/process-script.py /harddisk:/
  ```

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

  ```
  Router#script add process /harddisk:/ process-script.py
  Fri Aug 20 05:03:40.791 UTC
  process-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
---------------------------------------------------------------------------------------------
 process-script.py | process  | Config Checksum  | NEW         | Tue Aug 24 10:44:53 2021
=============================================================================================
```

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

Configure Checksum for Process Script

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 acts as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a warning message is displayed.

It is mandatory to configure the checksum to run the script.

Note Process scripts support the SHA256 checksum hash.

Before you begin

Ensure that the script is added to the script management repository. See Download the Script to the Router, on page 197.

Step 1 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/process/process-script.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
/harddisk:/mirror/script-mgmt/process/process-script.py
```

Make note of the checksum value.

Step 2 View the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:04:13.539 UTC
=============================================================================================
 Name                    | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
process-script.py        | process  | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
---------------------------------------------------------------------------------------------
 Script Name       : process-script.py
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The `Status` shows that the checksum is not configured.

Step 3 Configure the checksum.

Example:

```
Router#configure
Router(config)#script process process-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Tue Aug 20 05:10:10.546 UTC
Router(config)#end
```

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:15:17.296 UTC
=============================================================================================
 Name                   | Type     | Status      | Last Action | Action Time
---------------------------------------------------------------------------------------------
 process-script.py      | process  | Ready       | NEW         | Fri Aug 20 05:20:41 2021
---------------------------------------------------------------------------------------------
 Script Name       : process-script.py
 Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:20:41 2021
      Checksum     : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script fails. It is mandatory for the checksum values to match for the script to run.

Register the Process Script as an Application

Register the process script with the app manager to enable the script. The registration is mandatory for using process script on the router.

Before you begin

Ensure that the following prerequisites are met before you register the script:

- Download the Script to the Router, on page 197
- Configure Checksum for Process Script, on page 217

Step 1 Register the script with an application (instance) name in the app manager.

Example:

```
Router#configure
Fri Aug 20 06:10:19.284 UTC
Router(config)#appmgr process-script my-process-app
Router(config-process)#executable process-script.py
```

Here, `my-process-app` is the application for the executable `process-script.py` script.

Step 2 Provide the arguments for the script.

Example:

```
Router(config-process)#run-args --host <host-name> --runtime 3 --log script
```

Step 3 Set a restart policy for the script if there is an error.

Example:

```
Router(config-process)#restart on-failure max-retries 3
Router(config-process)#commit
```

Here, the maximum attempts to restart the script is set to 3. After 3 attempts, the script stops.

You can set more options to restart the process:

Keyword	Description
always	Always restart automatically. If the process exits, a scheduler queues the script and restarts the script. **Note** This is the default restart policy.
never	Never restart automatically. If the process exits, the script is not rerun unless you provide an action command to invoke the process.
on-failure	Restart on failure automatically. If the script exits successfully, the script is not scheduled again.
unless-errored	Restart script automatically unless errored.
unless-stopped	Restart script automatically unless stopped by the user using an action command.

Step 4 View the status of the registered script.

Example:

```
Router#show appmgr process-script-table
Fri Aug 20 06:15:44.244 UTC
Name            Executable         Activated    Status     Restart Policy   Config Pending
```

```
--------------- ------------------ --------- ------------- ---------------- --------------
my-process-app  process-script.py      No      Not Started  On Failure             No
```

The script is registered but is not active.

Activate the Process Script

Activate the process script that you registered with the app manager.

Before you begin

Ensure that the following prerequisites are met before you run the script:

- Download the Script to the Router, on page 197
- Configure Checksum for Process Script, on page 217
- Register the Process Script as an Application, on page 218

Step 1 Activate the process script.

Example:

```
Router#appmgr process-script activate name my-process-app
Fri Aug 20 06:20:55.006 UTC
```

The instance `my-process-app` is activated for the process script.

Step 2 View the status of the activated script.

Example:

```
Router#show appmgr process-script-table
Fri Aug 20 06:22:03.201 UTC
Name             Executable         Activated    Status     Restart Policy   Config Pending
---------------  ------------------ --------- ------------- ---------------- --------------
my-process-app   process-script.py      Yes       Running    On Failure             No
```

The process script is activated and running.

Note You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes are pending. The status of the modification is indicated in the `Config Pending` option. In the example, value `No` indicates that there are no configuration changes that must be activated.

Obtain Operational Data and Logs

Retrieve the operational data and logs of the script.

Before you begin

Ensure that the following prerequisites are met before you obtain the operational data:

- Download the Script to the Router, on page 197
- Configure Checksum for Process Script, on page 217
- Register the Process Script as an Application, on page 218
- Activate the Process Script, on page 220

Step 1 View the registration information, pending configuration, execution information, and run time of the process script.

Example:

```
Router#show appmgr process-script my-process-app info
Fri Aug 20 06:20:21.947 UTC
Application: my-process-app

  Registration info:
    Executable                    : process-script.py
    Run arguments                 : --host <host-name> --runtime 3 --log script
    Restart policy                : On Failure
    Maximum restarts              : 3

  Pending Configuration:
    Run arguments                 : --host <host-name> --runtime 3 --log script
    Restart policy                : Always

  Execution info and status:
    Activated                     : Yes
    Status                        : Running
    Executable Checksum           : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b

    Last started time             : Fri Aug 20 06:20:21.947
    Restarts since last activate  : 0/3
    Log location                  :
/harddisk:/mirror/script-mgmt/logs/process-script.py_process_my-process-app
    Last exit code                : 1
```

Step 2 View the logs for the process scripts. App manager shows the logs for errors and output.

Example:

The following example shows the output logs:

```
Router#show appmgr process-script my-process-app logs output
Fri Aug 20 06:25:20.912 UTC
[2021-08-20 06:20:55,609] INFO [sample-process]:: Beginning execution of process..
[2021-08-20 06:20:55,609] INFO [sample-process]:: Connecting to host '<host-name>'
[2021-08-20 06:20:56,610] INFO [sample-process]:: Reading database..
[2021-08-20 06:20:58,609] INFO [sample-process]:: Listening for requests..
```

The following example shows the error logs with errors:

```
Router#show appmgr process-script my-process-app logs errors
Fri Aug 20 06:30:20.912 UTC
----------Run ID:1632914459  Fri Aug 20 06:30:20 2021----------
Traceback (most recent call last):
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 121, in <module>
    main(args)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 97, in main
    printer()
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 37, in wrapper
    result = func(*args, **kwargs)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 88, in printer
```

```
    time.sleep(1)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 30, in _handle_timeout
    raise TimeoutError(error_message)
__main__.TimeoutError: Timer expired
----------Run ID:1632914460  Fri Aug 20 06:31:03 2021----------
```

This example shows the log without errors:

```
Router#show appmgr process-script my-process-app logs errors
Fri Aug 20 06:30:20.912 UTC
----------Run ID:1624346220  Fri Aug 20 10:46:44 2021----------
----------Run ID:1624346221  Fri Aug 20 10:47:50 2021----------
----------Run ID:1624346222  Fri Aug 20 10:52:39 2021----------
----------Run ID:1624346223  Fri Aug 20 10:53:45 2021----------
----------Run ID:1624346224  Fri Aug 20 11:07:17 2021----------
----------Run ID:1624346225  Fri Aug 20 11:08:23 2021----------
----------Run ID:1624346226  Fri Aug 20 11:09:29 2021----------
----------Run ID:1624346227  Fri Aug 20 11:10:35 2021----------
----------Run ID:1624346228  Fri Aug 20 11:11:41 2021----------
```

Managing Actions on Process Script

The process script runs as a daemon continuously. You can, however, perform the following actions on the process script and its application:

Table 51: Feature History Table

Action	Description
Deactivate	Clears all the resources that the application uses. `Router#appmgr process-script deactivate name my-process-app` You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes do not take effect.
Kill	Terminates the script if the option to stop the script is unresponsive. `Router#appmgr process-script kill name my-process-app`
Restart	Restarts the process script. `Router#appmgr process-script restart name my-process-app`
Start	Starts an application that is already registered and activated with the app manager. `Router#appmgr process-script start name my-process-app`
Stop	Stops an application that is already registered, activated, and is currently running. Only the application is stopped; resources that the application uses is not cleared. `Router#appmgr process-script stop name my-process-app`

Example: Check CPU Utilization at Regular Intervals Using Process Script

In this example, you use the process script to check CPU utilization at regular intervals. The script does the following actions:

- Monitor the CPU threshold value.
- If the threshold value equals or exceeds the value passed as argument to the script, log an error message that the threshold value has exceeded.

Before you begin

Ensure you have completed the following prerequisites before you register and activate the script:

1. Create a process script `cpu-utilization-process.py`. Store the script on an external server or copy the script to the harddisk of the router.

```
import time
import os
import xmltodict
import re
import argparse

from cisco.script_mgmt import xrlog
from iosxr.netconf.netconf_lib import NetconfClient

log = xrlog.getScriptLogger('Sample')
syslog = xrlog.getSysLogger('Sample')

def cpu_memory_check(threshold):
    """
    Check total routes in router
    """
    filter_string = """
    <system-monitoring xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-wdsysmon-fd-oper">
      <cpu-utilization>
        <node-name>0/RP0/CPU0</node-name>
          <total-cpu-one-minute/>
      </cpu-utilization>
    </system-monitoring>"""
    nc = NetconfClient(debug=True)
    nc.connect()
    do_get(nc, filter=filter_string)
    ret_dict = _xml_to_dict(nc.reply, 'system-monitoring')
    total_cpu =
int(ret_dict['system-monitoring']['cpu-utilization']['total-cpu-one-minute'])
    if total_cpu >= threshold:
        syslog.error("CPU utilization is %s, threshold value is %s"
%(str(total_cpu),str(threshold)))
    nc.close()

def _xml_to_dict(xml_output, xml_tag=None):
    """
    convert netconf rpc request to dict
    :param xml_output:
    :return:
    """
```

```
    if xml_tag:
        pattern = "<data>\s+(<%s.*</%s>).*</data>" % (xml_tag, xml_tag)
    else:
        pattern = "(<data>.*</data>)"
    xml_output = xml_output.replace('\n', ' ')
    xml_data_match = re.search(pattern, xml_output)
    ret_dict = xmltodict.parse(xml_data_match.group(1))
    return ret_dict

def do_get(nc, filter=None, path=None):
    try:
        if path is not None:
            nc.rpc.get(file=path)
        elif filter is not None:
            nc.rpc.get(request=filter)
        else:
            return False
    except Exception as e:
        return False
    return True

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("threshold", help="cpu utilization threshold",type=int)
    args = parser.parse_args()
    threshold = args.threshold
    while(1):
        cpu_memory_check(threshold)
        time.sleep(30)
```

Configure the script with the desired threshold criteria. This default threshold is configured to alert when CPU utilization exceeds this value. The script checks the CPU utilization every 30 seconds.

2. Add the script from the external server or harddisk to the script management repository. See Download the Script to the Router, on page 197.
3. Configure the checksum to verify the authenticity and integrity of the script. See Configure Checksum for Process Script, on page 217.

Step 1 Register the process script `cpu-utilization-process.py` with an instance name `my-process-app` in the app manager.

Example:

```
Router(config)#appmgr process-script my-process-app
Router(config-process)#executable cpu-utilization-process.py
Router(config-process)#run-args <threshold-value>
```

Step 2 Activate the registered application.

Example:

```
Router(config-process)#appmgr process-script activate name my-process-app
```

Step 3 Check the script status.

Example:

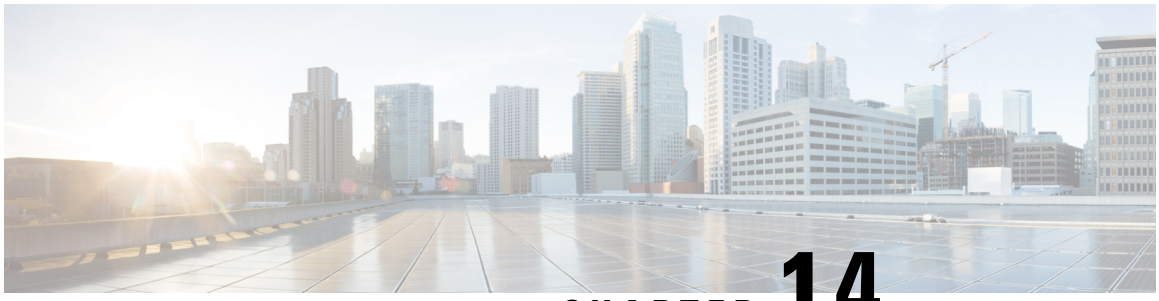
```
Router#show appmgr process-script-table
Thu Sep 30 18:15:03.201 UTC
Name             Executable                  Activated    Status     Restart Policy   Config Pending
---------------  --------------------------  ---------  ------------ ---------------  --------------
my-process-app   cpu-utilization-process.py      Yes       Running    On Failure            No
```

Step 4 View the log.

Example:

```
Router#show appmgr process-script my-process-app logs errors
RP/0/RP0/CPU0:Sep 30 18:03:54.391 UTC: python3_xr[68378]: %OS-SCRIPT_MGMT-3-ERROR :
Script-test_process: CPU utilization is 6, threshold value is 5
```

An error message is displayed that the CPU utilization has exceeded the configured threshold value, and helps you take corrective actions.



CHAPTER 14

EEM Scripts

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:

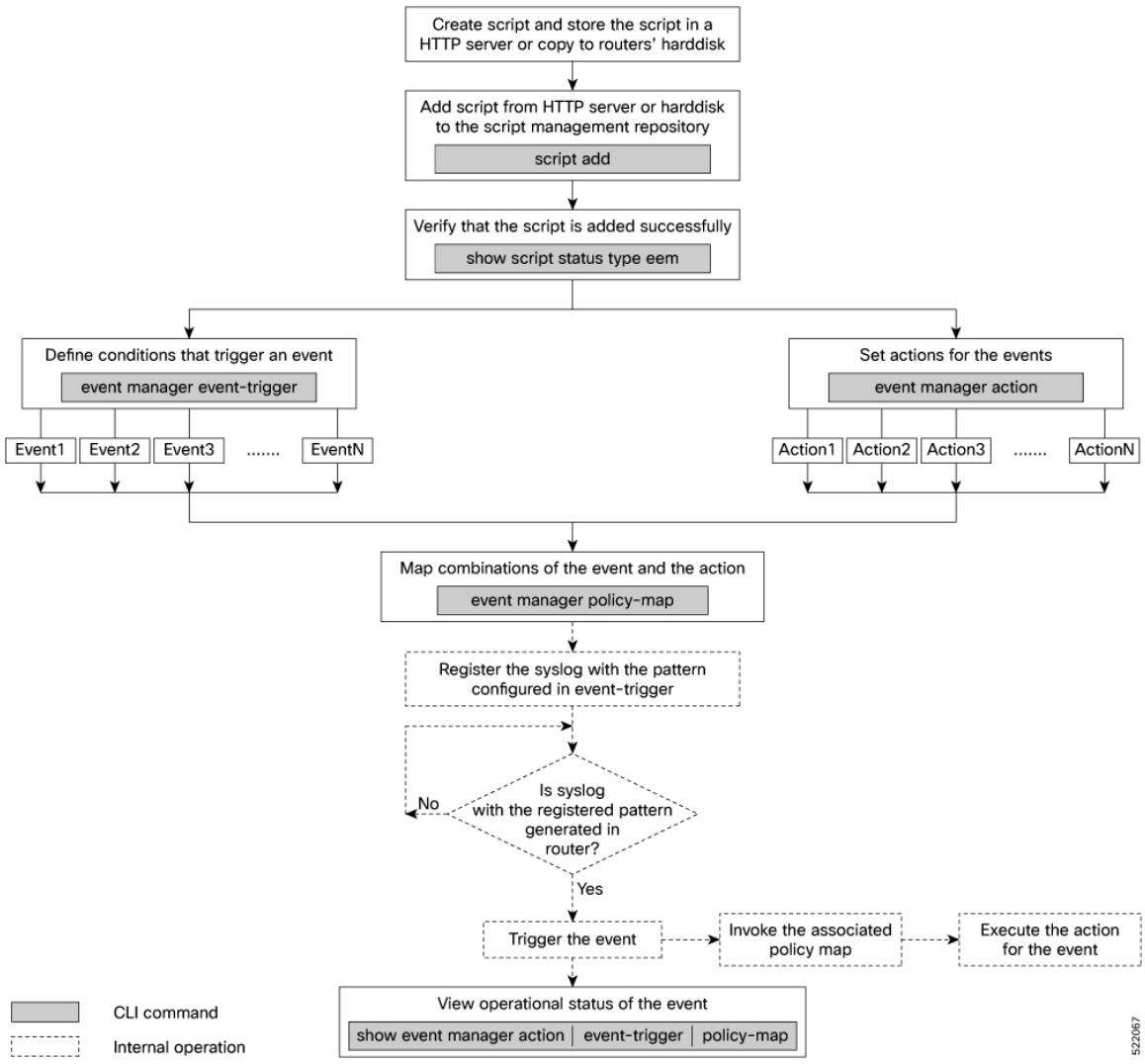
- Download the script—Store the eem script on an HTTP server or copy to the harddisk of the router. Add the eem script from the HTTP server or harddisk to the script management repository on the router using the **script add eem** command.

- 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.

Keyword	Description

type	Configure the type of event.
	Note In Cisco IOS XR Release 7.3.2, you can configure only syslog events.
	• 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 (`UP`, `DOWN`, or `ANY`). If event-trigger is configured for track state `UP`, then it gets triggered when the track state changes from `DOWN` to `UP`, 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.
	`Router#event manager telemetry sensor-path <sensor-path> json-query <query>`
	It is mandatory to enable model-driven telemetry using the command:
	`Router#telemetry model-driven`

Step 2 Configure the type for the event.

- Syslog:

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

For syslog, set the pattern to match. In this example, the pattern `L2-BM-6-ACTIVE` is the match value. If a syslog is generated on the router with a pattern that matches this configured pattern, the event gets triggered.

- Timer:

Watchdog timer—

```
Router(config)#event manager event-trigger  <event-name>
 type timer watchdog value <countdown-timer-value-in-seconds>
```

Cron timer—

```
Router(config)#event manager event-trigger <event-name>
 type timer cron cron-entry "<cron string>"
```

- Track:

```
Router(config)#event manager event-trigger <event-name>
 type track name <track-name> status {up | down | any}
```

- Telemetry:

Match criteria as exact-match—

```
Router(config)#event manager event-trigger <event-name>
 query json-path <query> match-criteria exact-match value <value>
 type telemetry sensor-path <telemetry-sensor-path>
sample-interval <sample-interval-in-seconds>
```

Match criteria as threshold—

```
Router(config)#event manager event-trigger <event-name> query
json-path <query> match-criteria threshold {equal-to | greater-equal-to |
greater-than | less-equal-to | less-than| not-equal-to} <value>
type telemetry sensor-path <telemetry-sensor-path> sample-interval <sample-interval-in-seconds>
```

Match criteria as rate—

```
Router(config)#event manager event-trigger <event-name>
 query json-path <query> match-criteria rate direction {any | decreasing | increasing}
value {equal-to| greater-equal-to | greater-than | less-equal-to | less-than | not-equal-to}
<value>
 type telemetry sensor-path <telemetry-sensor-path> sample-interval <sample-interval-in-seconds>>
```

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 231

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

Table 52: Feature History Table

Feature Name	Release Information	Description
Add Multiple Events In a Policy Map With a Single EEM Script	Release 7.5.1	With this feature, 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. Using EEM scripts, you can create a logical correlation of events in the policy map and configure multiple actions for detectors such as timer, object-tracking, and telemetry events via sensor path.

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 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 231
- Create Actions for Events, on page 234

Step 1 Create a policy map.

Example:

```
Router(config)#event manager policy-map policy1
```

```
Router(config)#event manager policy-map policy1
 trigger multi-event ["(<event1> AND <event2>) AND (<event3> OR <event4>)" |
 occurrence <count> | period <time in seconds>]
```

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 or multiple events 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 231
- Create Actions for Events, on page 234
- Create a Policy Map of Events and Actions, on page 235

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.

```
Router#show event manager event-trigger event1 detailed
Fri Nov 19 04:21:45.558 UTC

Event trigger name: event1
Event esid: 107
Event type: timer
Event occurrence: NA
Event period: NA
Event rate-limit: NA
Event triggered count: 12861
Event policy reg count: 1
Event status: active
Timer type: watchdog
Timer value: 10

Policy mapping info
1    event1                     policy1
```

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.

```
Router#show event manager policy-map policy1 all detailed
Fri Nov 19 11:35:40.282 UTC

Policy name: policy1
Policy occurrence: 3
Policy period: 120
Policy triggered count: 0
Policy status: active
Multi event policy: FALSE

Events mapped to the policy
No.       Name                        Status
1         event2                      active

Actions mapped to the policy
No.       Name                        Checksum
1         action1                     SHA256
```

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

```
Router#show event manager policy-map policy1 detailed
Fri Nov 19 11:05:38.828 UTC

Policy name: policy1
Policy occurrence: 2
Policy period: 60
Policy triggered count: 0
Policy status: active
Multi event policy: TRUE
Multi event string : "event1 OR (event4 AND event2)"
Current Correlation State : FALSE

Events mapped to the policy
No.       Name                          Status         Corr Status    Reset time(sec)
1         event1                        active         0              0
2         event2                        active         0              0
3         event4                        active         0              0

Actions mapped to the policy
No.       Name                          Checksum
1         action2                       SHA256
```

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')

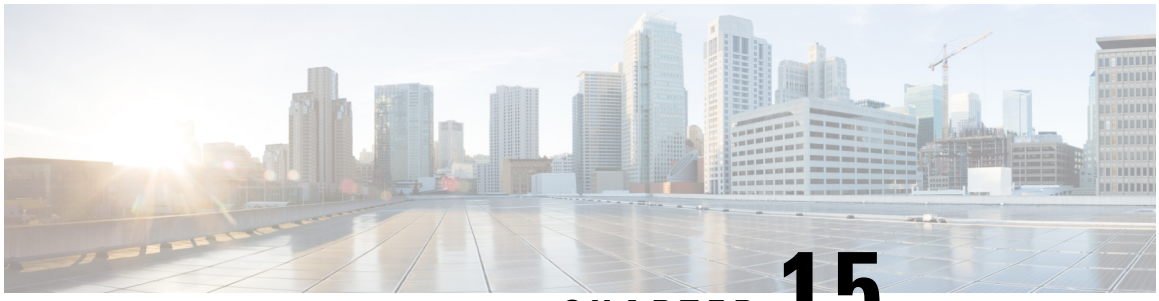
  syslog.info('Execution of event manager action script event_script_action_bundle_shut.py ended')
```

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"
!
```



CHAPTER 15

Model-Driven Command-Line Interface

This section shows the CLI commands that are based on YANG data models and can be used on the router console.

Model-Driven CLI to Display Data Model Structure

Table 53: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Show YANG Operational Data	Release 7.3.2	This feature enables you to use a traditional CLI command to display YANG data model structures on the router console and also obtain operational data from the router in JSON or XML formats. The functionality helps you transition smoothly between CLI and YANG models, easing data retrieval from your router and network. This feature introduces the **show yang operational** command.

Cisco IOS XR Software provides a rich set of show commands and data models to access data from the router and network. The show commands present unstructured data, whereas data models are structured data that can be encoded in XML or JSON formats. However, both the access points do not always present the same view. Network operators who work on show commands face challenges with adopting the data models when transitioning to programmatic interfaces.

With this feature, these adoption challenges are overcome using **show yang operational** command that is driven by data models. The command uses the data model as the base to display the structured data using traditional CLI command. Using this command, you can simplify parsing scripts via XML and JSON formats.

A data model has a structured hierarchy: model, module, container, and leaf. The following example shows the structure of `ietf-interfaces.yang` data model:

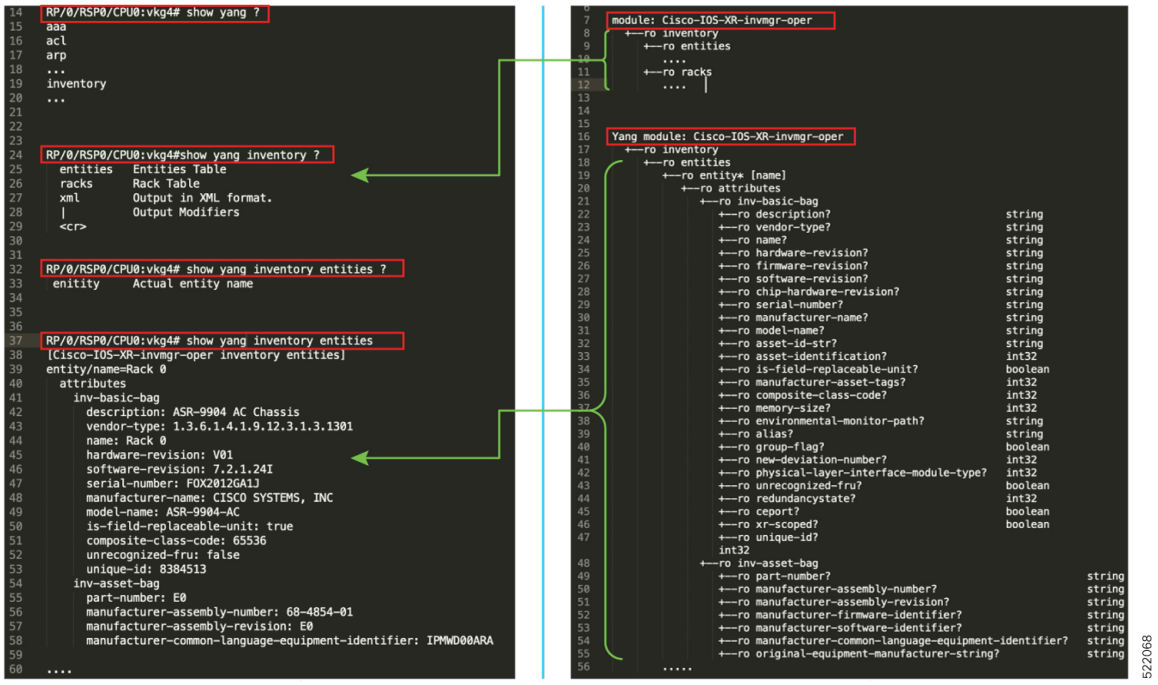
```
ietf-interfaces.yang
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                        string
  |     +--rw description?                string
  |     +--rw type                        identityref
  |     +--rw enabled?                    boolean
  |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
  +--ro interfaces-state
        +--ro interface* [name]
        +--ro name               string
        +--ro type               identityref
        +--ro admin-status       enumeration {if-mib}
```

In the example, the hierarchy of the data model is as follows:

- Model—ietf-interfaces.yang
- Module—ietf-interfaces
- Container—interfaces, interface-state
- Node—interface* [name]
- Leaf—name, description, type, enabled, link-up-down-trap-enable, admin-status

You can use the **show yang operational** command to navigate to the leaf level as you do in a data model.

The image show a mapping between CLI and data model, and how the structured data is displayed on the console.



The table shows various queries that can be used to navigate through the hierarchy of a data model using the CLI command. The queries are demonstrated using `Cisco-IOS-XR-interfaces-oper.yang` data model as an example.

Operational Query	Description

Search specific top-level nodes	Search and produce the output of keywords from top-level nodes.
	`Router#show yang operational`
	`Router#show yang operational | include <component>`
	The following example shows the search result for interfaces:
	`Router#show yang operational | include interface`
	`Wed Jul  7 00:02:37.982 PDT`
	`drivers-media-eth-oper:ethernet-interface`
	`ifmgr-oper:interface-dampening`
	`ifmgr-oper:interface-properties`
	`interface-cem-oper:cem`
	`l2vpn-oper:generic-interface-list-v2`
	`pfi-im-cmd-oper:interfaces`

All the instances of the container	Lists all the models at the root level container and its container name.
	`Router#show yang operational ?`
	You can also see the containers for a partially typed keyword. For example, keyword search for `mpls-` displays all the containers with `mpls` :
	`Router#show yang operational mpls-`
	`mpls-io-oper-mpls-ea  mpls-io-oper-mpls-ma`
	`mpls-ldp-mldp-oper:mpls-mldp`
	`mpls-lsd-oper:mpls-lsd  mpls-lsp-oper:mpls-lsd-nodes`
	`mpls-ldp-mldp-oper:mpls-mldp`
	`mpls-vpn-oper:l3vpn  mpls-te-oper:mpls-tp`
	`mpls-te-oper:mpls-te`
	View the container data. The output of the command is in-line with the structure of the data model.
	`Router#show yang operational mpls-static-oper:mpls-static`
	`Request datatree:`
	`filter`
	`mpls-static (ka)`
	`{`
	`"Cisco-IOS-XR-mpls-static-oper:mpls-static": {`
	`"vrfs": {`
	`"vrf": [`
	`{`
	`"vrf-name": "default"`
	`}`
	`]`
	`},`
	`"summary": {`
	`"lsp-count": 0,`
	`"label-count": 0,`
	`"label-error-count": 0,`
	`"label-discrepancy-count": 0,`
	`"vrf-count": 1,`
	`"active-vrf-count": 1,`
	`"interface-count": 0,`
	`"interface-forward-reference-count": 0,`
	`"lsd-connected": true,`
	`"ribv4-connected": false,`
	`"ribv6-connected": false`
	`}`
	`}`
	`}`

Operational Query	Description

All the nodes of the container	`Router#show yang operational mpls-static-oper:mpls-static ?`
	`JSON Output in JSON format`
	`XML Output in XML format`
	`local-labels`
	`summary`
	`vrfs`
	`| Output Modifiers`
	`<cr>`
	Output in JSON Format:
	`Router#show yang operational man-netconf-oper:netconf-yang clients JSON`
	`Mon Sep 27 11:38:27.158 PST`
	`Request datatree:`
	`filter`
	`netconf-yang (ka)`
	`clients`
	`{`
	`"Cisco-IOS-XR-man-netconf-oper:netconf-yang": {`
	`"clients": {`
	`"client": [`
	`{`
	`"session-id": "1396267443",`
	`"version": "1.1",`
	`"connect-time": "52436839",`
	`"last-op-time": "1545",`
	`"last-op-type": "get",`
	`"locked": "No"`
	`}`
	`]`
	`}`
	`}`
	`}`
	Output in XML Format:
	`Router#show yang operational man-netconf-oper:netconf-yang clients XML`
	`Mon Sep 27 11:38:34.218 PST`
	`Request datatree:`
	`filter`
	`netconf-yang (ka)`
	`clients`
	`<netconf-yang`
	`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-man-netconf-oper">`
	`<clients>`
	`<client>`
	`<session-id>1396267443</session-id>`
	`<version>1.1</version>`
	`<connect-time>52443884</connect-time>`
	`<last-op-time>1545</last-op-time>`
	`<last-op-type>get</last-op-type>`
	`<locked>No</locked>`
	`</client>`
	`</clients>`
	`</netconf-yang>`

Operational Query	Description

Navigate until the last leaf level	`Router#show yang operational mpls-static-oper:mpls-static summary ?`
	`JSON Output in JSON format`
	`XML Output in XML format`
	`active-vrf-count`
	`im-connected`
	`interface-count`
	`interface-forward-reference-count`
	`mpls-enbled-interface-count`
	`vrf-count`
	`| Output Modifiers`
	`<cr>`
	View data specific to the leaf value. The `read only` (ro) leaves in a YANG model are considered as the state data (operational).
	`Router#show yang operational mpls-static-oper:mpls-static summary active-vrf-count`
	`Request datatree:`
	`filter`
	`mpls-static (ka)`
	`summary`
	`active-vrf-count`
	`{`
	`"Cisco-IOS-XR-mpls-static-oper:mpls-static": {`
	`"summary": {`
	`"active-vrf-count": [`
	`}`
	`}`
	`}`

Model-Driven CLI to Display Running Configuration in XML and JSON Formats

Table 54: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 7.3.2	This feature enables you to display the configuration data for Cisco IOS XR platforms in both JSON and XML formats. This feature introduces the **show run | [xml | json]** command.

The **show run | [xml | json]** command uses native, OpenConfig and unified models to retrieve and display data.

Use the following variations of the command to generate output:

- **show run | [xml | json]**—Shows configuration in YANG XML or JSON tree.
- **show run | [xml | json] openconfig**—Shows configuration in OpenConfig YANG XML tree.

- **show run | [xml | json] unified**—Shows configuration in unified model YANG XML tree.
- **show run** *component* **| [xml | json]**—Shows configuration in YANG XML or JSON tree for the top-level component. For example, **show run interface | xml**
- **show run** *component* **| [xml | json] unified**—Shows configuration in unified model YANG XML or JSON tree for the top-level component. For example, **show run interface | json unified**
- **show run** *component subcomponent* **| [xml | json]**—Shows configuration in YANG XML or JSON tree for the granular-level component. For example, **show run router bgp 12 neighbor 12.12.12.12 | xml**
- **show run** *component subcomponent* **| [xml | json] unified**—Shows configuration in unified model YANG XML or JSON tree for the granular-level component. For example, **show run router bgp 12 neighbor 12.12.12.12 | json unified**

XML Output

```
Router#show run | xml
Building configuration...
 <data>
  <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
   <interface-configuration>
    <active>act</active>
    <interface-name>GigabitEthernet0/0/0/0</interface-name>
    <shutdown></shutdown>
   </interface-configuration>
   <interface-configuration>
    <active>act</active>
    <interface-name>GigabitEthernet0/0/0/1</interface-name>
    <shutdown></shutdown>
   </interface-configuration>
   <interface-configuration>
    <active>act</active>
    <interface-name>GigabitEthernet0/0/0/2</interface-name>
    <shutdown></shutdown>
   </interface-configuration>
  </interface-configurations>
  <interfaces xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-interface-cfg">
   <interface>
    <interface-name>GigabitEthernet0/0/0/0</interface-name>
    <shutdown/>
   </interface>
   <interface>
    <interface-name>GigabitEthernet0/0/0/1</interface-name>
    <shutdown/>
   </interface>
   <interface>
    <interface-name>GigabitEthernet0/0/0/2</interface-name>
    <shutdown/>
   </interface>
  </interfaces>
 </data>
```

JSON Output

```
Router#show run | json
Building configuration...
{
  "data": {
    "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
       "interface-configuration": [
   {
      "active": "act",
```

```
      "interface-name": "GigabitEthernet0/0/0/0",
      "shutdown": [
          null
             ]
           },
           {
      "active": "act",
      "interface-name": "GigabitEthernet0/0/0/1",
      "shutdown": [
          null
      ]
     },
     {
      "active": "act",
      "interface-name": "GigabitEthernet0/0/0/2",
      "shutdown": [
          null
      ]
     }
   ],
 "Cisco-IOS-XR-man-netconf-cfg:netconf-yang": {
    "agent": {
      "ssh": true
     }
 },
 }
```

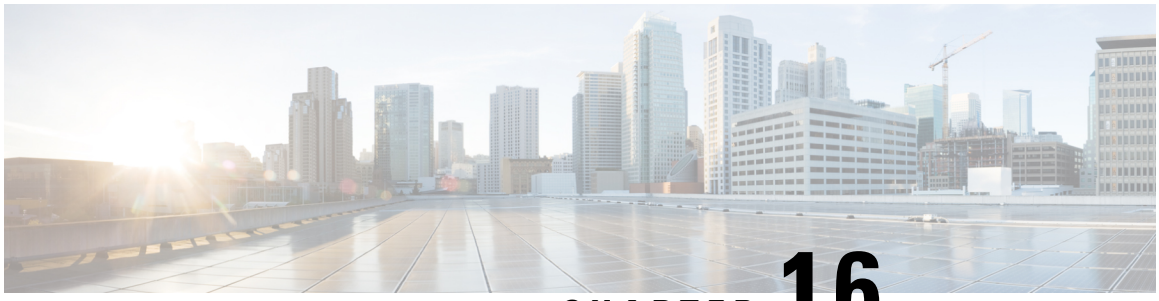
Granular-Level Component Output

```
Router#sh run router bgp 12 neighbor 12.12.12.12 | json unified
{
 "data": {
  "Cisco-IOS-XR-um-router-bgp-cfg:router": {
   "bgp": {
    "as": [
     {
      "as-number": 12,
      "neighbors": {
       "neighbor": [
        {
         "neighbor-address": "12.12.12.12",
         "remote-as": 12,
         "address-families": {
          "address-family": [
           {
            "af-name": "ipv4-unicast"
           }
          ]
         }
        }
       ]
      }
     }
    ]
   }
  }
 }
}
```

Unified Model Output

```
Router#sh run router bgp 12 | xml unified
<data>
 <router xmlns=http://cisco.com/ns/yang/Cisco-IOS-XR-um-router-bgp-cfg>
  <bgp>
```

```
    <as>
     <as-number>12</as-number>
     <bgp>
      <router-id>1.1.1.1</router-id>
     </bgp>
     <address-families>
      <address-family>
       <af-name>ipv4-unicast</af-name>
      </address-family>
     </address-families>
     <neighbors>
      <neighbor>
       <neighbor-address>12.12.12.12</neighbor-address>
       <remote-as>12</remote-as>
       <address-families>
        <address-family>
         <af-name>ipv4-unicast</af-name>
        </address-family>
       </address-families>
      </neighbor>
     </neighbors>
    </as>
   </bgp>
  </router>
</data>
```



CHAPTER 16

Manage Automation Scripts Using YANG RPCs

Table 55: Feature History Table

Feature Name	Release Information	Description
Manage Automation Scripts Using YANG RPCs	Release 7.3.2	This feature enables you to use remote procedure calls (RPCs) on YANG data models to perform the same automated operations as CLIs, such as edit configurations or retrieve router information.

You can use automation scripts to interact with the router using NETCONF, helper modules or gNMI python modules.

An SSH session must be established between the client and the server to run RPCs on a device. 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. To enable the NETCONF SSH agent, use the following commands:

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

After a NETCONF session is established, the client sends one or more RPC requests to the server. The server processes the requests and sends an RPC response back to the client. For example, the get-config operation retrieves the configuration of the device and the edit-config operation edits the configuration on the device.

For more information about data models and how to use the models

- Manage Common Script Actions Using YANG RPCs, on page 250
- Manage Exec Scripts Using RPCs, on page 252
- Manage EEM Script Using RPCs, on page 256

Manage Common Script Actions Using YANG RPCs

Table 56: Feature History Table

Feature Name	Release Information	Description
Manage Common Script Actions Using YANG RPCs	Release 7.5.1	This feature enables you to use YANG remote procedure calls (RPCs) on `Cisco-IOS-XR-infra-script-mgmt-act.yang` data model to perform actions on the automation scripts such as add or remove script from the script repository, run, or stop script from running.

This section provides information about YANG RPC messages for common actions on automation scripts. The `Cisco-IOS-XR-infra-script-mgmt-act.yang` action YANG model is enhanced to perform the actions such as adding or removing a script from the repository, and also include output responses. The output response provides a description about the action and displays the status as `True` for a successful action, and `False` for a failed action.

The YANG RPC supports these scripts:

- Config
- Exec
- Process
- EEM

The following section shows the various script actions, sample RPC request, and RPC response.

Add Script

You can add up to a maximum of 10 scripts to the script repository. You can set the script type to config, exec, process, or eem. The following example shows the RPC to add the exec script to the repository:

```
<add xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<vrf></vrf>
<source>/harddisk:/</source>
<script-name>sample.py</script-name>
</add>
```

You can add more than one script to the repository simultaneously:

```
<add xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<source>/harddisk:/</source>
<script-name>sample2.py</script-name>
<script-name>sample3.py</script-name>
</add>
```

To add a checksum value to the script, use the following RPC request:

```
<add-checksum xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<source>/harddisk:/</source>
<script-checksums>
    <script-name>sample.py</script-name>
<checksum>e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855</checksum>
</script-checksums>
</add-checksum>
```

You can add more than one script with their checksum values:

```
<add-checksum xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<source>/harddisk:/</source>
<script-checksums>
    <script-name>sample.py</script-name>
<checksum>e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855</checksum>
</script-checksums>
<script-checksums>
    <script-name>sample2.py</script-name>
<checksum>e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855</checksum>
</script-checksums>
</add-checksum>
```

Remove Script

To remove script from the repository, provide the script type and the script name. You can send an RPC request to remove up to 10 scripts.

```
<remove xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<script-name>sample.py</script-name>
</remove>
```

You can remove more than one script simultaneously:

```
<remove xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<script-type>exec</script-type>
<script-name>sample2.py</script-name>
<script-name>sample3.py</script-name>
</remove>
```

The following example shows a sample RPC response indicating that the script `sample1.py` is removed from the repository:

```
<responses>
<script-name>sample.py<script-name>
 <response>sample.py has been removed from the script repository</response>
<status>True</status>
<responses>
```

Stop Script

You must provide the request ID for the script instance to be stopped.

```
<stop xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
<request-id>1622058854</request-id>
<description></description>
</stop>
```

The following example shows that the script has stopped:

```
<script-stop-response>
           <response></response>
```

```
            <status>True</status>
</script-stop-response>
```

Run Script

You must provide the script name to run the script. You can also configure the log levels to one of these values—Critical, Debug, Error, Info, or Warning.

```
<run xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
    <script-name>sample.py</script-name>
    <argument-list></ argument-list>
    <description></description>
    <log-level></log-level>
    <log-path></log-path>
    <max-runtime></max-runtime>
</run>
```

The following example shows a sample RPC response where the script with the request ID `1622058854` is run:

```
<script-run-response>
 <response>Script run scheduled</response>
 <request-id>1622058854</request-id>
 <status>True</status>
</script-run-response>
```

Manage Exec Scripts Using RPCs

The following data models support exec scripts:

- Edit or get configuration—Cisco-IOS-XR-infra-script-mgmt-cfg.yang
- Perform action—Cisco-IOS-XR-infra-script-mgmt-act.yang
- Retrieve operational data—Cisco-IOS-XR-infra-script-mgmt-oper.yang

This section provides examples of using RPC messages on exec scripts, and also the YANG data model and equivalent CLI command to perform the tasks:

Add Script

You use data model to add an exec script from an external repository to the `harddisk:/mirror/script-mgmt/exec` script management repository on the router.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-act.yang	**script add exec** *script-location script.py* See.

RPC Request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <script-add-type-source xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
    <type>exec</type>
    <source>/harddisk:</source>
    <file-name-1>sample1.py</file-name-1>
  </script-add-type-source>
</rpc>
```

Syslog:

```
Router: script_manager[66762]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_manager: sample1.py has been added to the script repository
```

Configure Checksum

Every script is associated with a checksum value for integrity. You can configure the checksum using data models.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-act.yang	**script exec** *sample1.py* **checksum SHA256** *checksum-value* See, .

RPC Request:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:16fa22ed-3f46-4369-806a-3bccd1aefcaf">
  <nc:edit-config>
    <nc:target>
      <nc:candidate/>
    </nc:target>
    <nc:config>
      <scripts xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-cfg">
        <exec-script>
          <scripts>
            <script>
              <script-name>sample1.py</script-name>
              <checksum>
                <checksum-type>sha256</checksum-type>

<checksum>5103a843032505decc37ff21089336e4bcc6a1061341056ca8add3ac5d6620ef</checksum>
              </checksum>
            </script>
          </scripts>
        </exec-script>
      </scripts>
    </nc:config>
  </nc:edit-config>
</nc:rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:16fa22ed-3f46-4369-806a-3bccd1aefcaf"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Run Script

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-act.yang	**script run** *sample1.py*

RPC Request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <script-run xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
    <name>sample1.py</name>
  </script-run>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:d54247c7-cf29-42f2-bfb8-517d6458f77c" xmlns="urn:ietf:
params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Syslog:

```
Router: UTC: script_control_cli[67858]: %OS-SCRIPT_MGMT-6-INFO : Script-control:
Script run scheduled: sample1.py. Request ID: 1631795207
Router: script_agent_main[248]: %OS-SCRIPT_MGMT-6-INFO : Script-script_agent: Script
execution sample1.py (exec) Started : Request ID : 1631795207 :: PID: 18710
```

Stop Script

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-act.yang	**script stop** *value [short-decription]*

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <script-stop-request xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
    <request>1614930988</request>
  </script-stop-request>
</rpc>
```

Remove Script

You can remove scripts from the script management repository. The data about script management and execution history is not deleted when the script is removed.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-act.yang	**script remove exec** *script.py* See,.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <script-remove-type xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
    <type>exec</type>
    <file-name-1>load_modules_ut.py</file-name-1>
  </script-remove-type>
</rpc>
```

Show Script Execution

View the status of the script execution.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-infra-script-mgmt-oper.yang	**show script execution** *[request-id <value>] [name <filename>] [status {Exception | Executed | Killed | Started | Stopped | Timed-out}] [reverse] [last <number>]*

RPC Request:

```
-------------- Sent to NETCONF Agent ---------------
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:7fd0d184-0004-4a51-9765-d29bc94c793b">
  <get>
    <filter>
      <script xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-oper">
        <execution>
          <requests>
            <request>
              <request-id>1631795207</request-id>
              <detail>
                <execution-detail/>
              </detail>
            </request>
          </requests>
        </execution>
      </script>
    </filter>
  </get>
</rpc>
```

RPC Response:

```
----------------- Received from NETCONF agent --------------------
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:7fd0d184-0004-4a51-9765-d29bc94c793b"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <script xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-oper">
      <execution>
        <requests>
          <request>
            <request-id>1631795207</request-id>
            <detail>
              <execution-detail>
                <execution-summary>
                  <request-id>1631795207</request-id>
                  <return-val>0</return-val>
                  <script-type>exec</script-type>
                  <script-name>sample1.py</script-name>
                  <duration>60.65s</duration>
                  <event-time>Thu Sep 16 12:26:46 2021</event-time>
                  <status>Executed</status>
                </execution-summary>
                <execution-detail>

<log-path>/harddisk:/mirror/script-mgmt/logs/sample1.py_exec_1631795207</log-path>
                  <run-options>Logging level - INFO, Max. Runtime - 300s, Mode -
Background</run-options>
                </execution-detail>
                <execution-event>
                  <description>None</description>
                  <duration>0.00s</duration>
```

```
                  <event>New</event>
                  <time>Thu Sep 16 12:26:46 2021</time>
                </execution-event>
                <execution-event>
                  <description>Script execution started. PID (18710)</description>
                  <duration>0.03s</duration>
                  <event>Started</event>
                  <time>Thu Sep 16 12:26:46 2021</time>
                </execution-event>
                <execution-event>
                  <description>Script execution complete</description>
                  <duration>60.65s</duration>
                  <event>Executed</event>
                  <time>Thu Sep 16 12:27:47 2021</time>
                </execution-event>
              </execution-detail>
            </detail>
          </request>
        </requests>
      </execution>
    </script>
  </data>
</rpc-reply>
```

Manage EEM Script Using RPCs

The following data model supports eem scripts:

- Edit configuration—Cisco-IOS-XR-um-event-manager-policy-map-cfg.yang

The model is augmented to `Cisco-IOS-XR-um-event-manager-cfg.yang` data model.

This section provides examples of using RPC messages on eem scripts, and also the YANG data model and equivalent CLI command to perform the tasks:

Define Actions for Events Using Data Model

You use data model to create actions for events.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-um-event-manager-policy-map-cfg	**event manager event-trigger** *event-name* **occurance** *value* **period seconds** *value* **period seconds** *value***type syslog pattern** *"syslog-pattern"* **severity** *syslog-severity* See **event manager action** *action-name* **username** *username* **type script script-name** *python-script-name.py* **maxrun seconds** *value* **checksum sha256** *checksum-value* See.

RPC Request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
   <target>
    <candidate/>
   </target>
   <config>
    <event xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-event-manager-cfg">
     <manager>
      <event-trigger
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-event-manager-policy-map-cfg">
       <event>
        <event-name>event_1</event-name>
        <occurrence>2</occurrence>
        <period>
         <seconds>60</seconds>
        </period>
        <type>
         <syslog>
          <pattern>"Syslog for EEM script"</pattern>
          <severity>
           <warning/>
          </severity>
         </syslog>
        </type>
       </event>
      </event-trigger>
   <actions xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-event-manager-policy-map-cfg">

      <action>
       <action-name>action_1</action-name>
        <type>
         <script>
          <script-name>event_script_1.py</script-name>
          <maxrun>
           <seconds>30</seconds>
          </maxrun>
          <checksum>
         <sha256>bb19a7a286db72aa7c7bd75ad5f224eea1062b7cdaaeee06f11f0f86f976831d</sha256>
```

```
          </checksum>
         </script>
        </type>
       <username>eem_user_1</username>
      </action>
     </actions>
    </manager>
   </event>
  </config>
 </edit-config>
</rpc>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
<commit>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:16fa22ed-3f46-4369-806a-3bccd1aefcaf"
xmlns="urn:ietf:params:xml:ns:
netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Create Policy Map for Events and Actions Using Data Model

You use data model to create actions for events.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-um-event-manager-policy-map-cfg	**event manager policy-map** *policy-name* **action** *action-name* **trigger event** *event-name* See, .

RPC Request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <edit-config>
 <target>
  <candidate/>
  </target>
 <config>
 <event xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-event-manager-cfg">
 <manager>
  <policy-maps xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-event-manager-policy-map-cfg">

     <policy-map>
      <policy-map-name>policy_1</policy-map-name>
      <trigger>
       <event>event_1</event>
      </trigger>
      <actions>
      <action>
        <action-name>action_1</action-name>
      </action>
     </actions>
    </policy-map>
   </policy-maps>
   </manager>
```

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

<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
  <commit/>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:16fa22ed-3f46-4369-806a-3bccd1aefcaf"
xmlns="urn:ietf:params:xml:ns:
netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Operational Model for EEM Script

Table 57: Feature History Table

Feature Name	Release Information	Description
Operational Data Model for EEM Script	Release 7.5.2	You can programmatically retrieve the operational status of events, actions, and policy maps using the YANG data model. In earlier releases, you used the `show event manager` command to view the operational status of event scripts. This release introduces `Cisco-IOS-XR-ha-eem-policy-oper.yang` and `Cisco-IOS-XR-event-manager-policy-map-oper.yang` data models.

Operational Data Model to Retrieve Actions

You use data model to view the details of an action. IOS XR actions are RPC statements that trigger an operation or execute a command on the router. This 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.

YANG Data Model	Equivalent Command
Cisco-IOS-XR-ha-eem-policy-oper	**show event manager action** *action-name* **detailed** See, View Operational Status of Event Scripts, on page 236.

RPC Request:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:62b9e81b-5d9e-44f6-8a5d-d193a0f8b3d3">
  <get>
    <filter>
```

```
      <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
        <action-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
          <action-name>
            <action-name>action2</action-name>
          </action-name>
        </action-names>
      </eem>
    </filter>
  </get>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:62b9e81b-5d9e-44f6-8a5d-d193a0f8b3d3"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
      <action-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
        <action-name>
          <action-name>action2</action-name>
          <action-name-xr>action2</action-name-xr>
          <script-name>event_script_2.py</script-name>
          <action-type>script</action-type>
          <triggered-count>7</triggered-count>
          <policy-count>1</policy-count>
          <max-run>20</max-run>
          <checksum-enabled>SHA256</checksum-enabled>
          <last-run-status>Success</last-run-status>
          <user-name>eem_user</user-name>

<checksum-string>270b9730e77c9bd6f5784084ed21e29d8d7b8edaf8f98a4513879a1631c493ad</checksum-string>

          <action-policy-map>
            <policy-name>policy3</policy-name>
          </action-policy-map>
        </action-name>
      </action-names>
    </eem>
  </data>
</rpc-reply>
```

Operational Data Model to Retrieve Policy Map

You use data model to view the details of a policy map.

YANG Data Model	Equivalent Command
Cisco-IOS-XR-ha-eem-policy-oper	**show event manager policy-map** *policy-name* **detailed** See, View Operational Status of Event Scripts, on page 236.

RPC Request:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:3cec3f3a-395b-4763-b1a1-1053149da60c">
  <get>
    <filter>
```

```
      <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
        <policy-map-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
          <policy-map-name>
            <policy-name>policy4</policy-name>
          </policy-map-name>
        </policy-map-names>
      </eem>
    </filter>
  </get>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:3cec3f3a-395b-4763-b1a1-1053149da60c"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
      <policy-map-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
       <policy-map-name>
         <policy-name>policy4</policy-name>
          <policy-name-xr>policy4</policy-name-xr>
          <policy-status>active</policy-status>
          <policy-occurrence>2</policy-occurrence>
          <policy-period>30</policy-period>
          <policy-triggered-count>0</policy-triggered-count>
          <event-count>2</event-count>
          <action-count>1</action-count>
          <policy-event-map>
            <event-name>event5</event-name>
            <event-status>active</event-status>
            <corr-status>false</corr-status>
            <reset-time>0</reset-time>
          </policy-event-map>
          <policy-event-map>
            <event-name>event4</event-name>
            <event-status>active</event-status>
            <corr-status>false</corr-status>
            <reset-time>0</reset-time>
          </policy-event-map>
          <policy-action-map>
            <action-name>action4</action-name>
            <checksum-enabled>SHA256</checksum-enabled>
          </policy-action-map>
          <multi-event-policy>true</multi-event-policy>
          <current-correlation-state>false</current-correlation-state>
          <multi-event-string>&amp;quot;event4 AND event5&amp;quot;</multi-event-string>
        </policy-map-name>
      </policy-map-names>
    </eem>
  </data>
 </rpc-reply>
```

Operational Data Model to Retrieve Events With Trigger Conditions

You use data model to view the details of a event-trigger conditions.

YANG Data Model	Equivalent CLI
Cisco-IOS-XR-ha-eem-policy-oper	show event manager event-trigger *event-trigger-name* **detailed** See, View Operational Status of Event Scripts, on page 236.

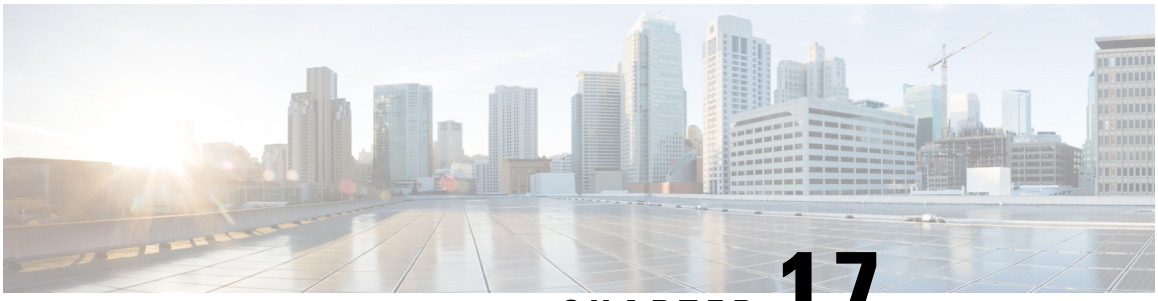
RPC Request:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:77229832-1a44-47e4-b0cf-2c2066ac579a"><nc:get>
    <filter>
      <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
        <event-trigger-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
          <event-trigger-name>
            <event-name>event4</event-name>
          </event-trigger-name>
        </event-trigger-names>
      </eem>
    </filter>
  </get>
</rpc>
```

RPC Response:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:77229832-1a44-47e4-b0cf-2c2066ac579a"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <eem xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ha-eem-policy-oper">
      <event-trigger-names
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-event-manager-policy-map-oper">
        <event-trigger-name>
          <event-name>event4</event-name>
          <event-name-xr>event4</event-name-xr>
          <event-status>active</event-status>
          <event-type>syslog</event-type>
          <eventesid>16</eventesid>
          <event-occurrence>NA</event-occurrence>
          <event-period>NA</event-period>
          <rate-limit>0</rate-limit>
          <event-triggered-count>2</event-triggered-count>
          <event-policy-reg-count>1</event-policy-reg-count>
          <event-policy-map>
            <policy-name>policy4</policy-name>
          </event-policy-map>
          <event-syslog-info>
            <pattern>%PKT_INFRA-LINK-3-UPDOWN : Interface GigabitEthernet0/0/0/4, changed
state to Down</pattern>
            <severity>ALL</severity>
          </event-syslog-info>
          <event-timer-info>
            <wd-info>
              <timer-value>0</timer-value>
            </wd-info>
          </event-timer-info>
          <event-telemetry-info>
            <sample-interval>0</sample-interval>
          </event-telemetry-info>
        </event-trigger-name>
```

```
      </event-trigger-names>
    </eem>
  </data>
</rpc-reply>
```



CHAPTER 17

Script Infrastructure and Sample Templates

Table 58: Feature History Table

Feature Name	Release Information	Description
Contextual Script Infrastructure	Release 7.3.2	When you create and run Python scripts on the router, this feature enables a contextual interaction between the scripts, the IOS XR software, and the external servers. This context, programmed in the script, uses Cisco IOS XR Python packages, modules, and libraries to: • obtain operational data from the router • set configurations and conditions • detect events in the network and trigger an appropriate action

You can create Python scripts and execute the scripts on routers running Cisco IOS XR software. The software supports the Python packages, libraries and dictionaries in the software image. For more informtion about the script types and to run the scripts using CLI commands To run the same actions using NETCONF RPCs,

Cisco IOS XR, Release 7.3.2 supports creating scripts using Python version 3.5.

Cisco IOS XR, Release 7.5.1 supports creating scripts using Python version 3.9.

Cisco IOS XR Python Packages

Table 59: Feature History Table

Feature Name	Release Information	Description
Upgraded IOS XR Python from Version 3.5 to Version 3.9	Release 7.5.1	This upgrade adds new modules and capabilities to create Python scripts and execute the scripts on routers running Cisco IOS XR software. Some of the modules added as part of the upgraded IOS XR Python 3.9 are: hashlib, idna, packaging, pyparsing, six, yaml.

With on-box Python scripting, automation scripts that was run from an external controller is now run on the router. To achieve this functionality, Cisco IOS XR software provides contextual support using SDK libraries and standard protocols.

The following Python third party application packages are supported by the scripting infrastructure and can be used to create automation scripts.

Package	Description	Support Introduced in Release
appdirs	Chooses the appropriate platform-specific directories for user data.	Release 7.3.2
array	Defines an object type that can compactly represent an array of basic values: characters, integers, floating point numbers.	Release 7.3.2
asn1crypto	Parses and serializes Abstract Syntax Notation One (ASN.1) data structures.	Release 7.3.2
chardet	Universal character encoding auto-detector.	Release 7.3.2
concurrent.futures	Provides a high-level interface for asynchronously executing callables.	Release 7.3.2
ecdsa	Implements Elliptic Curve Digital Signature Algorithm (ECDSA) cryptography library to create keypairs (signing key and verifying key), sign messages, and verify the signatures.	Release 7.3.2

Package	Description	Support Introduced in Release
enum	Enumerates symbolic names (members) bound to unique, constant values.	Release 7.3.2
email	Manages email messages.	Release 7.3.2
google.protobuf	Supports language-neutral, platform-neutral, extensible mechanism for serializing structured data.	Release 7.3.2
hashlib	Implements a common interface to many different secure hash and message digest algorithms.	Release 7.5.1
idna	Supports the Internationalized Domain Names in Applications (IDNA) protocol as specified in RFC 5891.	Release 7.5.1
ipaddress	Provides capability to create, manipulate and operate on IPv4 and IPv6 addresses and networks.	Release 7.3.2
jinja2	Supports adding functionality useful for templating environments.	Release 7.3.2
json	Provides a lightweight data interchange format.	Release 7.3.2
markupsafe	Implements a text object that escapes characters so it is safe to use in HTML and XML.	Release 7.3.2
netaddr	Enables system-independent network address manipulation and processing of Layer 3 network addresses.	Release 7.3.2
packaging	Add the necessary files and structure to create the package.	Release 7.5.1
pdb	Defines an interactive source code debugger for Python programs.	Release 7.3.2
pkg_resources	Provides runtime facilities for finding, introspecting, activating and using installed distributions.	Release 7.3.2

Package	Description	Support Introduced in Release
psutil	Provides library to retrieve information on running processes and system utilization such as CPU, memory, disks, sensors and processes.	Release 7.3.2
pyasn1	Provides a collection of ASN.1 modules expressed in form of pyasn1 classes. Includes protocols PDUs definition (SNMP, LDAP etc.) and various data structures (X.509, PKCS).	Release 7.3.2
pyparsing	Provides a library of classes to construct the grammar directly in Python code.	Release 7.5.1
requests	Allows sending HTTP/1.1 requests using Python.	Release 7.3.2
shellescape	Defines the function that returns a shell-escaped version of a Python string.	Release 7.3.2
six	Provides simple utilities for wrapping over differences between Python 2 and Python 3.	Release 7.5.1
subprocess	Spawns new processes, connects to input/output/error pipes, and obtain return codes.	Release 7.3.2
urllib3	HTTP client for Python.	Release 7.3.2
xmltodict	Makes working with XML feel like you are working with JSON.	Release 7.3.2
yaml	Provides a human-friendly format for structured data, that is both easy to write for humans and still parsable by computers.	Release 7.5.1

Cisco IOS XR Python Libraries

Cisco IOS XR software provides support for the following SDK libraries and standard protocols.

Library	Syntax

gnmi	To connect to gnmi client #
	`from iosxr.gnmi.gnmi_lib import GNMIClient`
	`gnmi = GNMIClient()`
	For more information, see Use Automation Scripts to Interact with the Router via gNMI RPCs, on page 273.

xrlog	`# To generate syslogs #`
	`from cisco.script_mgmt import xrlog`
	
	`syslog = xrlog.getSysLogger('template_exec')`

netconf	`#To connect to netconf client #`
	`from iosxr.netconf.netconf_lib import NetconfClient`
	
	`nc = NetconfClient(debug=True)`

xrclihelper	`# To run native xr cli and config commands`
	`from iosxr.xrcli.xrcli_helper import *`
	
	`helper = XrcliHelper(debug = True)`

config_validation	`# To validate configuration #` `import cisco.config_validation as xr`
eem	`# For EEM operations #` `from iosxr import eem`
precommit	`# For Precommit script operations #` `from cisco.script_mgmt import precommit`

Sample Script Templates

Table 60: Feature History Table

Feature Name	Release Information	Description
Github Repository for Automation Scripts	Release 7.5.1	You now have access to sample scripts and templates published on the Github repository. You can leverage these samples to use the python packages and libraries developed by Cisco to build your custom automation scripts for your network

Use these sample script templates based on script type to build your custom script.

To get familiar with IOS XR Python scripts, see the samples and templates on the Cisco Devnet developer program and Github repository.

Follow these instructions to download the sample scripts from the Github repository to your router, and run the scripts:

1. Clone the Github repository.

```
$git clone https://github.com/CiscoDevNet/iosxr-ops.git
```

2. Copy the Python files to the router's harddisk or a remote repository.

Precommit Script

The following example shows the template for precommit scripts

```
from cisco.script_mgmt import precommit

def sample_method():
    """
    Method documentation
    """

    cfg = precommit.get_target_configs()
    # cfg = precommit.get_target_configs(format="sysdb") for target config in sysdb format

    # process and verify target configs here.

    precommit.config_warning("Print a warning message in commit report")
    precommit.config_error("Print an error message in commit report and abort commit
operation")


if __name__ == '__main__':

    sample_method()
```

Config Script

The following example shows a code snippet for config script. Use this snippet in your script to import the libraries required to validate configuration and also generate syslogs.

```
#Needed for config validation
import cisco.config_validation as xr

#Used for generating syslogs
from cisco.script_mgmt import xrlog
syslog = xrlog.getSysLogger('Add script name here')

def check_config(root):
    #Add config validations
    pass

xr.register_validate_callback([<Add config path here>],check_config)
```

Exec Script

Use this sample code snippet in your exec script to import Python libraries to connect to NETCONF client and also to generate syslogs.

```
#To connect to netconf client
from iosxr.netconf.netconf_lib import NetconfClient

#To generate syslogs
syslog = xrlog.getSysLogger('template_exec')

def test_exec():
    """
```

```
    Testcase for exec script
    """
    nc = NetconfClient(debug=True)
    nc.connect()
    #Netconf or processing operations
    nc.close()


if __name__ == '__main__':
    test_exec()
```

Process Script

Use the following sample code snippet to trigger a process script and perform various actions on the script. You can leverage this snippet to create your own custom process script. Any exec script can be used as a process script.

```
To trigger script
Step 1: Add and configure script as shown in README.MD

Step 2: Register the application with Appmgr

Configuraton:
appmgr process-script my-process-app
executable test_process.py
run args --threshold <threshold-value>

Step 3: Activate the registered application
appmgr process-script activate name my-process-app

Step 4: Check script status
show appmgr process-script-table

Router#show appmgr process-script-table
Name             Executable         Activated    Status     Restart Policy   Config Pending
---------------  ------------------ ---------- ------------- ---------------- --------------
my-process-app   test_process.py      Yes       Running    On Failure            No

Step 5: More operations
Router#appmgr process-script ?
  activate    Activate process script
  deactivate  Deactivate process script
  kill        Kill process script
  restart     Restart process script
  start       Start process script
  stop        Stop process script
"""

#To connect to netconf client
from iosxr.netconf.netconf_lib import NetconfClient

#To generate syslogs
syslog = xrlog.getSysLogger('template_exec')

def test_process():
    """
    Testcase for process script
    """
    nc = NetconfClient(debug=True)
    nc.connect()
    #Netconf or any other operations
    nc.close()
```

```
if __name__ == '__main__':
    test_process()
```

EEM Script

You can leverage the following sample code to import Python libraries to create your custom eem script and also generate syslogs.

```
Required configuration:
User and AAA configuration

event manager event-trigger <trigger-name>
type syslog pattern "PROC_RESTART_NAME"

event manager action <action-name>
username <user>
type script script-name <script-name> checksum sha256 <checksum>

event manager policy-map policy1
trigger event <trigger-name>
action <action-name>

To verify:
Check for syslog EVENT SCRIPT EXECUTED: User restarted <process-name>

"""
#Needed for eem operations
from iosxr import eem

#Used to generate syslogs
from cisco.script_mgmt import xrlog
syslog = xrlog.getSysLogger(<add your script name here>)

# event_dict consists of details of the event
rc, event_dict = eem.event_reqinfo()

#You can process the information as needed and take action for example: generate a syslog.
#Syslog type can be emergency, alert, critical, error, exception, warning, notification,
info, debug

syslog.info(<Add you syslog here>)
```

Use Automation Scripts to Interact with the Router via gNMI RPCs

Table 61: Feature History Table

Feature Name	Release Information	Description
Automation Scripts for gNMI RPCs	Release 7.5.2	You can create automation scripts to connect to the gRPC Network Management Interface (gNMI) server and interact with the router using gNMI services. Based on gNMI-defined RPCs, you can use the automation script to connect to the gNMI server, manage the configuration of network devices, and query the operational data.

gRPC Network Management Interface (gNMI) is developed by Google. gNMI provides the mechanism to install, manipulate, and delete the configuration of network devices, and also to view operational data. The content provided through gNMI can be modeled using YANG. The supported operations are based on the gNMI defined RPCs:

```
from iosxr.gnmi.gnmi_lib import GNMIClient
gnmi = GNMIClient()

#Connect
gnmi.connect()

#Capabilities
cap = gnmi.capabilities()

#Get
get = gnmi.get(get_request)

#Set
set = gnmi.set(set_request)

#Disconnect
gnmi.disconnect()
```

- **gNMI Capabilities RPC:** This RPC allows the client to retrieve the gNMI capabilities that is supported by the target (router). This allows the target to validate the service version that is implemented and retrieve the set of models that the target supports. The models can then be specified in subsequent RPCs to restrict the set of data that is utilized. The `CapabilityRequest` RPC returns a response `CapabilityResponse` RPC.
- **gNMI GET RPC:** This RPC specifies how to retrieve one or more of the configuration attributes, state attributes or all attributes associated with a supported mode from a date tree. A `GetRequest` RPC is sent from a client to the target to retrieve values from the data tree. A `GetResponse` RPC is sent in response to the request.
- **gNMI SET RPC:** This RPC specifies how to set one or more configurable attributes associated with a supported model. A `SetRequest` RPC is sent from a client to a target to update the values in the data tree. The actions contained in a `SetRequest` RPC is treated as a single transaction. If any element of the

transaction fails, the entire transaction fails and is rolled back. A `SetResponse` RPC is sent in response to the request.

- **gNMI Connect RPC:** This RPC specifies how to initiaize a connection to the client.
- **gNMI Disconnect RPC:** This RPC specifies how to end the connection with the client.

Restrictions for the gNMI Protocol

The following restrictions apply to the gNMI protocol:

- Subscribe RPC services are not supported.
- Only JSON_IETF encoding for GET and SET requests is supported
- CLI over GNMI is not supported

Follow the procedure to use automation scripts to interact with the router via gNMI services:

Step 1 Create script using the `GNMIClient` python module.

Example:

In this example, you create a script to connect with the router using gNMI capabilities.

```
from iosxr.gnmi.gnmi_lib import GNMIClient

gnmi = GNMIClient()
gnmi.connect()
print("Getting capabilities")
cap = gnmi.capabilities()
print("Get")
get_req = """
path: {
    elem: {
        name: "network-instances"
    }
    elem: {
        name: "network-instance"
        key: {
            key: "name"
            value: "vrf_1"
        }
    }
    origin: "openconfig-network-instance"
}
type: CONFIG
encoding: JSON_IETF
"""
get = gnmi.get(get_req)
print("Set")
set_req = """
prefix: <
    origin:"openconfig-interfaces"
>
update: <
path: <
            elem: <
                name: "interfaces"
            >
            elem: <
                name: "interface"
```

```
                key: <
                    key: "name"
                    value: "MgmtEth0/RP0/CPU0/0"
                >
            >
            elem: <
                name: "config"
            >
        >
        val: <
            json_ietf_val: '{"description":"Testing failover case: testrole200"}'
        >
>
"""
set = gnmi.set(set_req)
import pdb;pdb.set_trace()
```

Step 2 Configure gRPC.

Example:

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

Step 3 Copy the script to the router.

Step 4 Verify that the script is available on the router.

Example:

```
Router#show script status detail
Tue Apr 12 23:10:50.453 UTC
=============================================================================================
 Name                      | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
 gnmi-sample-script.py     | exec     | Config Checksum  | NEW         | Tue Apr 12 10:18:23 2021
 =============================================================================================
Script Name        : gnmi-sample-scripy.py
Checksum           : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Script Description : View gNMI capabilities
History:
 --------
 1.   Action       : NEW
      Time         : Tue Apr 12 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
Router(config)#exit
```

Step 5 Add the script to the script management repository.

Example:

```
Router#script add <type> <location> <name>
```

In this example, you add an Exec script `gnmi-sample-script.py` to the router.

```
Router#script add exec /harddisk\: gnmi-sample-scripy.py
Tue Apr 18 16:16:46.427 UTC
Copying script from /harddisk:/gnmi-sample-scripy.py
gnmi-sample-scripy.py has been added to the script repository
```

Step 6 Configure the checksum.

Example:

```
Router(config)#script <type> <name> checksum SHA 256 <checksum>
```

In this example, you configure the checksum for the Exec script `gnmi-sample-script.py` to the router.

Example:

```
Router(config)#script exec gnmi-sample-script.py checksum SHA 256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Router(config)#end
```

Step 7 Run the script.

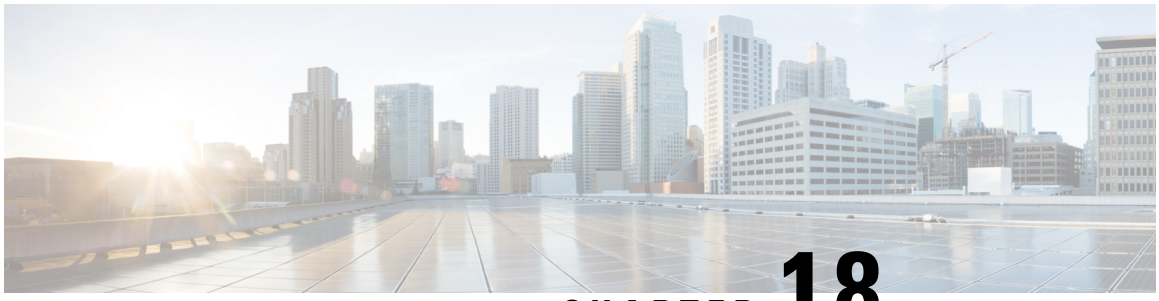
Example:

```
Router#script run gnmi-sample-script.py
Tue Apr 18 16:17:46.427 UTC
Script run scheduled: gnmi-sample-script.py. Request ID: 1634055439
Getting capabilities
.................................
```

The following example shows the output of the gNMI `get` operation:

```
notification: <
  timestamp: 1649917466577514766
  update: <
    path: <
      origin: "openconfig-interfaces"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface"
        key: <
          key: "name"
          value: "TenGigE0/0/0/0"
        >
      >
    >
    val: <
      json_ietf_val: "{\n \"config\": {\n  \"name\": \"TenGigE0/0/0/0\",\n  \"type\":
\"iana-if-type:ethernetCsmacd\",\n  \"enabled\": false\n },\n \"openconfig-if-ethernet:
ethernet\": {\n  \"config\": {\n   \"auto-negotiate\": false\n  }\n }\n}\n"
    >
  >
  update: <
    path: <
      origin: "openconfig-interfaces"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface"
        key: <
          key: "name"
          value: "TenGigE0/0/0/1"
        >
      >
    >
    val: <
      json_ietf_val: "{\n \"config\": {\n  \"name\": \"TenGigE0/0/0/1\",\n  \"type\":
\"iana-if-type:ethernetCsmacd\",\n  \"enabled\": false\n },\n \"openconfig-if-ethernet:
ethernet\": {\n  \"config\": {\n   \"auto-negotiate\": false\n  }\n }\n}\n"
```

```
    >
------------------------------- Output truncated for brevity ----------------------------------------
```



CHAPTER 18

Troubleshoot Automation Scripts

This chapter provides information about troubleshooting the automation scripts.

- Collect Debug Logs, on page 279

Collect Debug Logs

Table 62: Feature History Table

Feature Name	Release Information	Description
Debug Automation Scripts	Release 7.5.1	Use this feature to collect logs that contain debug information for ltraces and tech-support data. These logs aid in troubleshooting whenever the scripts are not working as expected. This feature introduces the **show tech-support script** command.

To automatically run **show** commands that display the debugging information specific to automation scripts, use the **show tech-support script** command in EXEC mode.

- **show version**
- **show platform**
- **show logging**
- **show running-config**
- **show install active**
- **show processes blocked location all**
- **show processes script_watcher_main location all**
- **show processes script_agent_main location all**
- **show processes checksum_verifier_main location all**

- **show memory summary location all**
- **show tech cfgmgr**
- **show tech eem**
- **show tech appmgr**
- **show script status detail**
- **show script execution detail**

In addition, the debug command collects the following data:

- All the script management log files in `/var/log` directory
- List all the files under `/pkg/lib/python3/` directroy
- Collects data about top processes consuming high CPU resources
- List all the processes initiated by the script manager
- Collect information about `/harddisk:/mirror/script-mgmt` directory

Run the debug command to collect information about the automation scripts (in zip format):

```
Router#show tech-support script
Mon Nov 15 23:28:46.849 UTC
++ Show tech start time: 2021-Nov-15.232847.UTC ++
Mon Nov 15 23:28:47 UTC 2021 Waiting for gathering to complete
.............................
Mon Nov 15 23:30:19 UTC 2021 Compressing show tech output
Show tech output available at
0/RP0/CPU0 : /harddisk:/showtech/showtech-script-2021-Nov-15.232847.UTC.tgz
++ Show tech end time: 2021-Nov-15.233019.UTC ++
```

View the collected debug zip files:

```
Router#dir harddisk:/showtech
Mon Nov 15 00:32:17.218 UTC

Directory of harddisk:/showtech
262146 -rw-rw-rw-. 1 1101085 Nov 15 23:24 showtech-script-2021-Nov-15.232322.UTC.tgz
262147 -rw-rw-rw-. 1 1143339 Nov 15 23:30 showtech-script-2021-Nov-15.232847.UTC.tgz

70553000 kbytes total (66887640 kbytes free)
```

Untar the collected zip file to view the list of debug log files:

```
Router#run
Mon Nov 15 00:32:29.724 UTC
[node0_RP0_CPU0:~]$cd /harddisk\:/showtech/
[node0_RP0_CPU0:/harddisk:/showtech]$ls -ltr
total 2196
-rw-rw-rw-. 1 root iosxr 1101085 Nov 15 23:24 showtech-script-2021-Nov-15.232322.UTC.tgz
-rw-rw-rw-. 1 root iosxr 1143339 Nov 15 23:30 showtech-script-2021-Nov-15.232847.UTC.tgz

[node0_RP0_CPU0:/harddisk:/showtech]$gunzip showtech-script-2021-Nov-15.232847.UTC.tgz
[node0_RP0_CPU0:/harddisk:/showtech]$ls -l
total 2612
-rw-rw-rw-. 1 root iosxr 1101085 Nov 15 23:24 showtech-script-2021-Nov-15.232322.UTC.tgz
-rw-rw-rw-. 1 root iosxr 1572864 Nov 15 23:30 showtech-script-2021-Nov-15.232847.UTC.tar
```

```
[node0_RP0_CPU0:/harddisk:/showtech]$tar -xvf showtech-script-2021-Nov-15.232847
showtech-script-2021-Nov-15.232847.UTC/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-ps-grep-python-output
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_action_log
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/config/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/logs/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/stderr.log
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/stdout.log
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/exec/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/exec/exec_sample_script.py
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/process/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/eem/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/.script-mgmt/
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/.script-mgmt/request_queue.json
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script-mgmt/.script-mgmt/script_db.json
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-ps-grep-python-output
showtech-script-2021-Nov-15.232847.UTC/cfg-node0_RP0_CPU0.tar
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_watcher_log
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-top-output-2
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0.tech.gz
showtech-script-2021-Nov-15.232847.UTC/system.tech.gz
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-top-output-2
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_agent_log
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/config/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/logs/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/stderr.log
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/logs/exec_sample_script.py_exec_1625009314/stdout.log
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/exec/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/exec/exec_sample_script.py
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/process/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/eem/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/.script-mgmt/
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/.script-mgmt/request_queue.json
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-script-mgmt/.script-mgmt/script_db.json
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0.tech.gz
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-top-output-1
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_control_log
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_manager_log
showtech-script-2021-Nov-15.232847.UTC/node0_RP1_CPU0-top-output-1
showtech-script-2021-Nov-15.232847.UTC/node0_RP0_CPU0-script_oper_log
```