Programmability Configuration Guide for Cisco 8000 Series Routers, Cisco IOS XR Releases

First Published: 2025-11-03

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

Preface

This cumulative guide provides a single, continuously updated version that includes all the latest IOS XR features and release updates. It simplifies your experience by letting you bookmark one link and access the complete guide, instead of navigating through multiple release-specific versions.

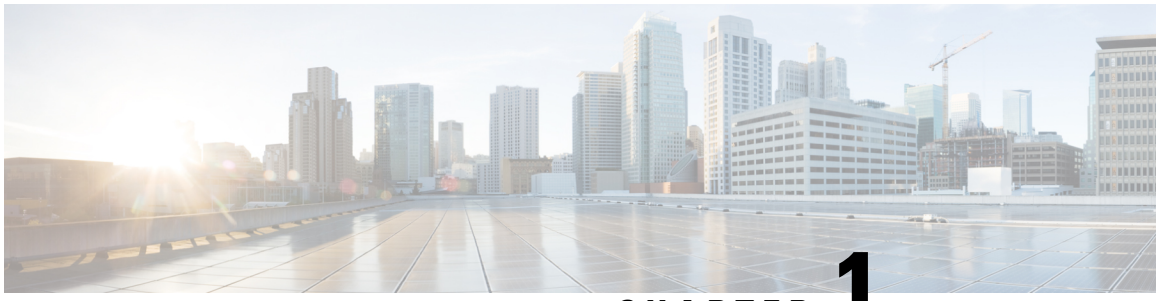
Specific changes or updates tied to individual releases are clearly called out within the relevant sections. For a list of features introduced in a specific release, refer to the Release Notes or the IOS XR Feature Finder.

The table lists the release numbers for which this document has been updated since its initial publication.

Table 1: Changes to this document

Date	Summary
October 2025	First published for Release 25.3.1

Preface



CHAPTER 1

YANG data models for programmability features

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*.

- Access data models, on page 1
- Get started with IOS XR YANG data models, on page 4

Access data models

You can access the data models using one of these options:

Access data models from router

To access data models directly from the router, you can use these steps:

Procedure

Step 1 Enter the global configuration mode.

Example:

```
Router#configure
```

Step 2 Configure the NETCONF network management protocol to remotely configure and manage the router using YANG data models.

Example:

```
Router(config)#netconf-yang agent ssh
```

Step 3 Commit the configuration.

Example:

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

Step 4 Establish a NETCONF session with the device and retrieve the capabilities information.

Example:

```
Router#show netconf-yang capabilities
Tue Sep 19 22:03:26.305 UTC
[Netconf capabilities]

   D: Has deviations

 Capability                                                                          | Revision |D
------------------------------------------------------------------------------------+----------+-
urn:ietf:params:netconf:base:1.1                                                    | -        |
urn:ietf:params:netconf:capability:candidate:1.0                                    | -        |
urn:ietf:params:netconf:capability:confirmed-commit:1.1                             | -        |
urn:ietf:params:netconf:capability:interleave:1.0                                   | -        |
urn:ietf:params:netconf:capability:notification:1.0                                 | -        |
urn:ietf:params:netconf:capability:rollback-on-error:1.0                            | -        |
urn:ietf:params:netconf:capability:validate:1.1                                     | -        |
http://cisco.com/ns/yang/Cisco-IOS-XR-8000-fib-platform-cfg                         |2019-04-05|
http://cisco.com/ns/yang/Cisco-IOS-XR-8000-lpts-oper                                |2022-05-05|
http://cisco.com/ns/yang/Cisco-IOS-XR-8000-platforms-npu-resources-oper             |2020-10-07|
http://cisco.com/ns/yang/Cisco-IOS-XR-8000-qos-oper                                 |2021-06-28|
http://cisco.com/ns/yang/Cisco-IOS-XR-Ethernet-SPAN-act                             |2021-03-22|
http://cisco.com/ns/yang/Cisco-IOS-XR-Ethernet-SPAN-cfg                             |2022-07-13|
http://cisco.com/ns/yang/Cisco-IOS-XR-Ethernet-SPAN-datatypes                       |2021-10-06|
http://cisco.com/ns/yang/Cisco-IOS-XR-Ethernet-SPAN-oper                            |2022-09-05|
http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-aaacore-cfg                               |2019-04-05|
http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-ldapd-cfg                                 |2022-06-22|
http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-ldapd-oper                                |2022-05-20|
http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-lib-cfg                                   |2020-10-22|
http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-lib-datatypes
---------------------------------   Truncated for brevity  -------------------------------------
```

By examining the capabilities, you can view the available data models for the software version installed on the router.

Access data models from Cisco Feature Navigator

To access data models from Cisco Feature Navigator, you can use these steps:

Procedure

Step 1 Go to Cisco Feature Navigator.

Step 2 If you have a Cisco.com account, click on the **Login** button and enter your credentials. If you don't have an account, you can click **Continue as Guest**.

You will be directed to the Cisco Feature Navigator main page.

Step 3 Click **YANG Data Models**.

Step 4 Select the **Product** and **Cisco IOS XR Release** based on your requirement.

The data models are listed based on type—Cisco XR native models, Unified models and OpenConfig models.

You can use the search field to search for specific data model of interest.

Step 5 Click the specific data model of interest to view more details.

The data model is displayed in a hierarchical tree structure making it easier to navigate and understand the relationships between different YANG modules, containers, leaves and leaf lists. You can apply filters to further narrow down the data model definitions for the selected platform and release based on status such as deprecated, obsolete and unsupported nodes.

You can also click the **Download** icon to export the data model information in Excel format.

This visual tree form helps you get insights into the nodes that you can use to automate your network.

The data models on Cisco Feature Navigator is regularly updated based on IOS XR release. If you encounter any problem or have suggestions for improvements, share your experience using Send us your feedback link.

Access data models from GitHub

To access the data models from GitHub repository, you can use these steps:

Procedure

Step 1 Go to the GitHub repository for data models.

On the repository page, you will find a list of folders based on IOS XR releases.

Step 2 Navigate to the release folder of interest to view the list of supported data models and their definitions. For example, if you want to access the data models for IOS XR release 7.10.1, click on the folder named `7.10.1`.

Inside the folder, you will find a list of YANG files representing different data models.

Step 3 Click on the YANG file you want to access to view its contents.

You can also click on the **Raw** button to see the raw code or use the **Download** button to download the file to your computer.

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. The unsupported sensor paths are documented as deviations. For example, `openconfig-acl.yang` provides details about the supported sensor paths, whereas `cisco-xr-openconfig-acl-deviations.yang` shows the unsupported sensor paths for `openconfig-acl.yang` model.

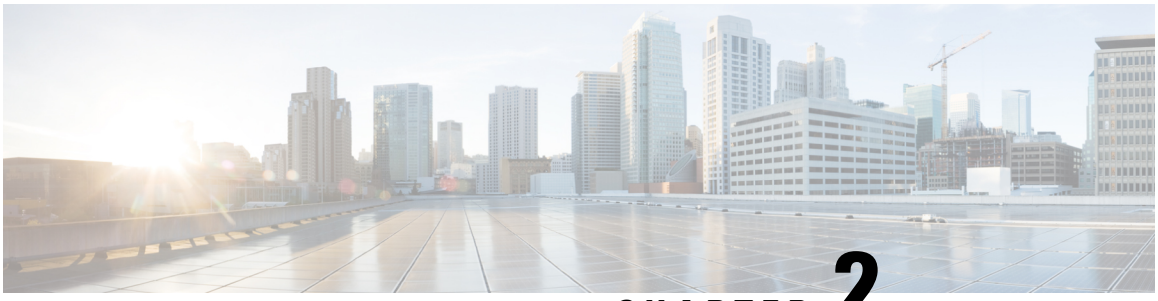
Step 4 Repeat the above steps for other versions or data models of interest.

The GitHub repository for IOS XR data models is regularly updated based on release. You can also contribute to the repository by submitting pull requests, opening issues if you encounter any problems or have suggestions for improvements.

Get started with IOS XR YANG data models

Here is a generic outline of the steps involved in programmatically configuring your router using YANG data models:

1. Enable network management protocol—Manage the router remotely using the protocols such as NETCONF or gRPC.
2. Install the necessary libraries and tools—Depending on the programming language you are using, you may need to install libraries or tools to programatically interact with the router. For example, if you are using Python, you might need to install the `ncclient` library.
3. Establish a session with the router—Use the programming language of your choice to establish a connection to the router using NETCONF or gRPC protocols. This involves providing connection parameters such as device IP address, username, password, and port number.
4. Retrieve the router capabilities—View the supported features and functionalities available on the router.
5. Create or modify configurations—Use YANG data models to create or modify the configuration on the router.
6. Apply the configuration—Push the updated configuration via the NETCONF or gRPC protocol to modify the router's running configuration to reflect the desired changes.
7. Validate the configuration—Verify that the changes are successfully applied. You can retrieve the running configuration or specific configuration parameters to ensure that the device is configured as intended.



CHAPTER 2

YANG data models

A YANG data model is a standardized network modeling language that

- defines the structure and constraints for configuration and operational data on network devices,
- enables automated setup and management of heterogeneous networks, and
- supports communication using protocols such as NETCONF and gRPC for scalable, consistent network operations.

Model-driven programmability

Model-driven programmability uses YANG data models to provide a flexible, rich framework for device automation. This framework offers multiple ways to interface with Cisco IOS XR devices via different transports, protocols, and encodings, which are decoupled from the data models for greater flexibility.

Data model layers

YANG data models organize device functions and configurations into logical layers. For example, one layer may define device interfaces, while another handles routing protocols. This separation simplifies automation and standardizes management across device types.

Protocol operations

YANG data models work in conjunction with protocols such as Network Configuration Protocol (NETCONF) and gRPC. These protocols use YANG models to access device capabilities, automate configuration, and retrieve operational data across the network.

Benefits of YANG data models

Configuring routers with YANG data models overcomes traditional limitations because these models:

- Provide a common structure for both configuration and operational data, and support NETCONF actions.
- Enable protocols to get, manipulate, and delete network device configuration.
- Automate management and operation of multiple routers throughout a heterogeneous network.

Additional information

Traditional CLI-based configuration is typically proprietary and highly text-based, making bulk configuration challenging in complex networks. By contrast, YANG data models use industry-standard languages to facilitate automation, interoperability, and operational consistency.

YANG data models

A YANG data model is a data modeling language specification that

- defines a standard, hierarchical structure for the configuration and operational data of network devices,
- enables robust network communication by specifying data relationships, constraints, actions, and notifications, and
- supports integration with network management protocols such as NETCONF and gRPC for automated configuration and monitoring.

- **YANG module**: A file or group of files that together define a single data model. Each module is uniquely identified by a namespace URL.
- **NETCONF/gRPC**: Protocols that use YANG data models for configuration and operational data exchange.

YANG data models must be obtained from the router. These models define a valid structure for the data exchanged between the router and the client, and are consumed by NETCONF and gRPC-enabled applications (gRPC supported only on 64-bit platforms). YANG models are categorized as follows:

- **Cisco-specific models**: Proprietary YANG models unique to Cisco devices. For details and representation, see Native models.
- **Common models (Open Config/OC)**: Industry-standard YANG models, typically from organizations such as IETF and IEEE. OC models have separate YANG modules for configuration data, operational data, and actions. See OC models for examples.

All YANG data models are stamped with semantic version 1.0.0 as the baseline from release 7.0.1 and later. For more details on YANG, refer to RFC 6020 and RFC 6087.

Data model requirements

YANG data models handle these types of requirements on routers (RFC 6244):

- **Configuration data**: A set of writable data required to transform a system from its initial state. For example, configuring entries in the IP routing table, or setting interface MTU or speed.
- **Operational state data**: Data obtained by the system at runtime, reflecting its operational status, which is typically transient and influenced by internal or external events (for example, OSPF routing entries).
- **Actions**: Supported via NETCONF actions to enable robust network-wide configuration transactions, ensuring atomic changes across devices.

For more information, see *RFC 6244*.

YANG model structure

YANG data models use a tree-based, hierarchical structure with nodes, making the models easy to understand and navigate. Each feature typically has a synthesized YANG model built from schemas. A model in tree format includes:

- Top level nodes and their subtrees
- Subtrees that augment nodes in other YANG models
- Custom RPCs

YANG node types

YANG defines these four node types for data modeling:

- **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 uniquely identified by one or more key leaves.
- **Container node**: Contains a grouping of related nodes (which may be any of the four types).

Each node is named and either defines a value or contains child nodes according to its type.

Components of a YANG module

A YANG module defines a single data model but can reference other modules or sub-modules. These standard statements are used:

- **Import**: Imports external modules.
- **Include**: Includes one or more sub-modules.
- **Augment**: Adds new definitions to another module at specified locations.
- **When**: Sets 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 the YANG module name, not the namespace.

Additional reference information

- All data models from release 7.0.1 and later are stamped with semantic version 1.0.0.
- For further reading on YANG, see RFC 6020 and RFC 6087.
- For industry-standard open YANG models, see the OpenConfig public repository.

Access data models

Cisco IOS XR routers ship with YANG files that define supported data models. You can use the NETCONF protocol and the ietf-netconf-monitoring request to view these models directly from the router.

Before you begin

- Ensure NETCONF is enabled on your router.
- Verify you have credentials for NETCONF client access.

Procedure

Step 1 Connect to your Cisco IOS XR router using a NETCONF client.

Step 2 Send an **ietf-netconf-monitoring** RPC request to retrieve the list of supported models.

Example:

Example RPC 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>
```

Step 3 Review the RPC response to see the available YANG models and details such as identifier, version, format, namespace, and location.

Example:

Example RPC response snippet:

```
<rpc-reply message-id="101" 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>openconfig-mpls-ldp</identifier>
<version>1.0.0</version>
<format>yang</format>
<namespace>http://openconfig.net/yang/ldp</namespace>
<location>NETCONF</location>
</schema>
<!-- Additional schema entries... -->
</schemas>
</netconf-state>
</data>
</rpc-reply>
```

YANG action

A YANG action is a network management operation that

- is defined in a YANG model using an RPC statement
- enables execution of specific commands or operations (such as ping or reload) on a network device via standardized management protocols like NETCONF or gRPC, and
- provides structured responses indicating the outcome of the requested operation.

YANG actions allow remote management systems to execute device-specific operations programmatically. Each action is modeled as an RPC (Remote Procedure Call) statement within a YANG module, making it accessible through automation tools and network controllers that communicate over NETCONF (using XML) or gRPC (using JSON). When an action request is received, the device executes the operation and returns a protocol-specific response. For example, common actions include "ping," "traceroute," "copy," and process restart commands.

Supported YANG actions and models

The following table lists common YANG actions and the associated YANG models. For a complete list of supported actions, refer to the YANG Data Models Navigator.

Table 2: YANG actions and models

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

PING NETCONF action

This example 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>
```

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>
```

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
```

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
```

Supported protocols and operational restrictions

- NETCONF actions use XML formatting; gRPC actions use JSON.
- Some actions—such as installing software—may have restrictions (for example, deprecated commands or maximum parameter limits).
- System admin models support only <get> and <get-config> operations; <edit-config> supports only merge. Delete, remove, and replace are not supported for system admin models.

YANG actions and data elements

A YANG action acts like a remote control button: when pressed (invoked through a protocol), the device performs the specified operation and returns the result, similar to how an ATM processes withdrawal commands remotely.

Configuration containers or leaves in YANG are not actions—they only represent data to store or retrieve, not executable operations.

YANG input validators and Get requests

A YANG input validator is a system component that:

- checks the validity of configuration data against OpenConfig YANG models
- verifies Get requests processed through protocols such as NETCONF or gNMI
- ensures that only explicitly configured OpenConfig leaves are returned in response to Get requests.

A Get request is an operation that:

- retrieves configuration or operational data from a network device,
- uses management protocols like NETCONF or gNMI
- returns either OpenConfig leaves or Cisco native model items based on system settings and operational modes.

YANG input validators and Get requests ensure configuration integrity and accurate state retrieval in Cisco IOS XR devices. Recent enhancements to these components result in stricter input validation, with OpenConfig models providing consistent handling of configuration queries. By default, Get requests now return only leaves explicitly configured via OpenConfig, improving data accuracy and compliance.

Legacy mode remains available for limited releases, preserving previous behaviors where Get requests could return all convertible Cisco native items.

Table 3: 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.

Legacy Mode Usage

- NETCONF: Add a legacy mode attribute to the **<get-config>** request **tag:get-config xmlns:xr-md="http://cisco.com/ns/yang/cisco-xr-metadata" xr-md:mode="legacy"**
- gNMI: Set the origin to **openconfig-legacy** in the request.

By default, OpenConfig leaves now return default values consistently using Explicit Basic Mode (see RFC6243).

YANG input validators usage guidelines and limitations

Use these best practices and follow the warnings when working with YANG input validators and OpenConfig in Cisco IOS XR:

Usage guidelines

Consider the following usage guidelines:

- After upgrading to Cisco IOS XR Release 7.10.1 or later, always commit OpenConfig changes to ensure OpenConfig leaves appear in Get requests.
- Use either gNMI or NETCONF as the management agent for OpenConfig, but do not use both simultaneously. The first successful commit determines the active agent.
- Fully configure each feature via OpenConfig, native model, or CLI. Items overridden in native models do not appear in the OpenConfig view.

Operational limitations

- Observe the same commitment requirements when downgrading or upgrading the software.
- The non-active agent can only configure native model items or perform Get requests. It cannot modify OpenConfig items until all OpenConfig leaves are first removed by the active agent.
- Changes made using CLI or Config Scripts are not reflected in OpenConfig system views.
- During commits, you can issue Get requests only on the running datastore. Edits, commits, or Get requests on candidate datastores belonging to other clients are rejected.

Warnings

- The command `show running-config | (xml | json) openconfig` displays the running OpenConfig configuration but cannot be filtered using XR CLI configuration keywords. Starting from Cisco IOS XR Release 2.4.4.1, this command is not supported.
- Do not use `load rollback changes` or `load commit changes` for rollbacks or commits that include OpenConfig leaves; these operations are not supported for OpenConfig data.
- System events, such as install operations or startup configuration failures, may remove configurations from the system, leaving OpenConfig views temporarily unsynchronized. Watch for syslog messages and reapply OpenConfig configurations when needed.
- Performing a `Commit Replace` operation through CLI removes all OpenConfig entries from the system.

Communication protocols

A communication protocol is a set of rules that

- establishes standardized exchanges of information between network devices,
- enables reliable and secure communication between clients and routers, and
- facilitates automation and programmable operations across different platforms.

In YANG-based systems, communication protocols connect the router and the client. These protocols enable clients to consume YANG data models, automating and programming network operations.

Supported protocols

YANG uses one of these protocols:

- Network Configuration Protocol (NETCONF)
- RPC framework (gRPC) by Google

Note gRPC is supported only on 64-bit platforms.

The transport and encoding mechanisms for these two protocols are shown in the table:

Table 4: Protocols and their supported transport and encoding or decoding mechanisms

Protocol	Transport	Encoding or Decoding
NETCONF	SSH	XML
gRPC	HTTP/2	JSON

NETCONF protocols

A NETCONF protocol is a network management protocol family that

- enables installation, modification, and deletion of device configuration data,
- uses XML-based data encoding for both configuration and protocol messages, and
- supports client-server communication through a simple RPC (Remote Procedure Call) mechanism.

NETCONF protocols provide standard tools for automating the management of network devices. Clients can programmatically update or retrieve configuration and operational data by exchanging XML-formatted messages with servers.

A NETCONF client issues an RPC call to configure a new interface, and the NETCONF server responds with an XML message confirming the change.

gRPC protocols

A gRPC protocol is a network communication framework that

- is based on Protocol Buffers (Protobuf), an open-source binary serialization format,
- enables flexible, efficient, and automated serialization of structured data for remote procedure calls (RPCs), and
- requires you to define message types using `.proto` files, where each message contains a series of name-value pairs for clear data structure.

Additional reference information: gRPC is open source and offers a lightweight alternative to traditional protocols like XML for transmitting data. Its design aims for both simplicity and high performance in exchanging data between systems.

Note gRPC is supported only on 64-bit platforms.

Examples:

- A network management system can use gRPC protocols to issue NETCONF RPCs and configure device features using data models.
- Developers use `.proto` files to define the messages and service methods that gRPC clients and servers exchange.
- REST APIs typically use JSON payloads and HTTP methods and do not rely on Protocol Buffers or `.proto` definitions as gRPC does.

Unified data models

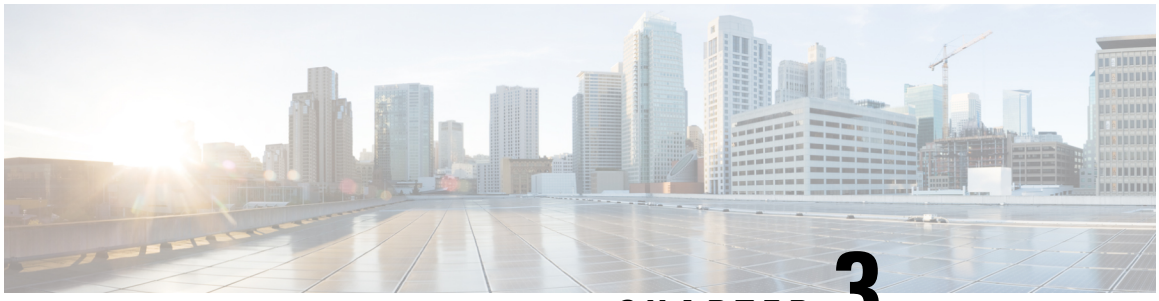
CLI-based YANG data models, also known as unified configuration models are introduced in Cisco IOS XR Software Release 7.0.1. The unified models provide a full coverage of the router functionality, and serves as a single abstraction for YANG and CLI commands. Unified models are generated from the CLI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



CHAPTER 3

Manage automation scripts using Yang RPCs

Automation scripts using YANG RPCs

An automation script is a software tool that

- interacts with network devices using standardized APIs such as NETCONF or gNMI,
- executes remote procedure calls (RPCs) defined by YANG data models to retrieve or edit device configuration, and
- enables automated, repeatable, and scalable management operations for network infrastructure.

To use automation scripts for remote management, you must establish an SSH session between the client (the script or application) and the network device (the server). For example, enabling the NETCONF SSH agent on a router requires these configuration commands:

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

Once connected, the client sends one or more RPC requests such as `get-config` to retrieve device configuration or `edit-config` to modify it. The server processes each request and sends a response.

Table 5: 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.

A Python script uses the NETCONF protocol to automatically update interface settings on multiple routers by sending `edit-config` RPCs. An automation tool retrieves system status from a network device using gNMI and YANG-defined telemetry models.

Common script actions

A common script action is a network automation operation that

- uses YANG remote procedure calls (RPCs) to manage automation scripts,
- supports multiple script types such as config, exec, process, and Embedded Event Manager (EEM), and
- produces output responses that describe the result of each action and indicate success or failure.

YANG RPCs (Remote Procedure Calls) enable centralized and programmatic management of automation scripts within a network device or system. The `Cisco-IOS-XR-infra-script-mgmt-act.yang` model allows users to add scripts to the repository, remove scripts, execute scripts, and stop running scripts. Each RPC provides a response that includes an operation status (`True` for success, `False` for failure) and a descriptive message.

Table 6: 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.

Add a script

Add scripts to the device script repository to automate configuration, execution, or monitoring processes. You can add up to 10 scripts at a time and specify optional checksum values for script integrity.

Use this procedure to add configuration, exec, process, or EEM scripts to your device repository. Adding scripts with checksums ensures the integrity and authenticity of scripts before execution.

Before you begin

Before you begin:

- Ensure you have access to the device with the required administrative privileges.
- Prepare your script files and confirm their locations.
- Optionally, compute and have ready the checksum value(s) for your scripts if you want to verify integrity.

Procedure

Step 1 To add a single script to the repository, use the following RPC call with the script type, source, and script name:

Example:

```
<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>
```

The specified script is added to the repository.

Step 2 To add multiple scripts at the same time, list each script name within the same RPC request:

Example:

```
<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>
```

All specified scripts are added to the repository in a single operation.

Step 3 To add a script with a checksum value for integrity validation, use the following RPC request:

Example:

```
<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>
```

The script is added to the repository with its checksum recorded for future validation.

Step 4 To add multiple scripts with their respective checksum values, include multiple `script-checksums` blocks:

Example:

```
<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>
```

All specified scripts are added with their checksums for verification.

Scripts are successfully added to the repository and are available for management or execution. Scripts with checksums are validated for integrity.

What to do next

After adding scripts, verify script availability and correct checksum status using the script management tools or commands available on your device.

Remove a script

Use this task to remove a script or multiple scripts from the repository by sending an RPC request with the script type and name.

Removing obsolete or unnecessary scripts helps maintain a clean and secure repository environment.

Perform this task when you need to delete scripts that are no longer required or must be replaced in the Cisco IOS XR script repository. This applies to both single and multiple script removals.

Before you begin

Before you begin, ensure you have the required access privileges to send RPC requests to the device and know the exact script type and script names you want to remove.

Procedure

Step 1 Provide the script type and script name(s) for each script to remove.

Example:

```
<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>
```

Step 2 If removing multiple scripts, include each script name within the request (up to 10).

Example:

```
<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>
```

Step 3 Send the RPC request to the device for processing.

Step 4 Verify the RPC response to ensure the scripts were removed successfully.

Example:

```
<responses>
            <script-name>sample.py</script-name>
            <response>sample.py has been removed from the script repository</response>
```

```
<status>True</status>
</responses>
```

The response indicates the status of each script removal.

The specified scripts are removed from the repository. Confirmation is provided in the RPC response.

What to do next

Review the list of scripts in the repository to confirm that only the required scripts remain.

Stop a script

Use this task to halt an active script on the device. This is useful if you need to interrupt a script that is currently running or if you need to recover from potential script-related issues.

Stopping a script requires the request ID assigned when the script was started. You must send an API request using the documented XML format to instruct the system to terminate the specific script instance.

Before you begin

Before you begin:

- Identify the request ID for the script you want to stop.
- Ensure you have access and permission to send API requests to the target device.

SUMMARY STEPS

1. Send a stop request using the required XML payload, replacing with the script's actual request ID.
2. Verify that the API response includes <status>True</status>, confirming the script has stopped.

DETAILED STEPS

Procedure

Step 1 Send a stop request using the required XML payload, replacing with the script's actual request ID.

Example:

```
<stop xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
          <request-id>1622058854</request-id>
          <description></description>
           </stop>
```

Step 2 Verify that the API response includes <status>True</status>, confirming the script has stopped.

Example:

```
<script-stop-response>
            <response></response>
            <status>True</status>
            </script-stop-response>
```

If the status is not True, confirm the request ID is correct and that the script is currently running.

The script has successfully stopped when status returns True.

The specified script instance is stopped and can no longer perform any operations.

What to do next

You may submit new scripts or manage script instances as needed now that the prior script has been terminated.

Run a script

Execute a specified script file on the device. Optionally, configure logging options for monitoring and troubleshooting.

Running a script automates management or operational tasks and may require different logging levels for troubleshooting. Perform this task when you need to automate a workflow, gather data, or perform custom operations via scripting.

Before you begin

Before you begin, ensure:

- The script file is uploaded and accessible on the device.
- You have the correct permissions to execute scripts.

Procedure

Step 1 Specify the script name to run.

Example:

```
<run xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-act">
          <script-name>sample.py</script-name>
          </run>
```

Step 2 (Optional) Set the desired logging level. Available levels include critical, debug, error, info, or warning.

Example:

```
<log-level>Info</log-level>
```

Step 3 (Optional) Define other parameters such as log path, maximum runtime, or add script arguments as required.

Example:

```
<argument-list>--verbose</argument-list>
          <log-path>/var/logs</log-path>
          <max-runtime>300</max-runtime>
```

Step 4 Submit the run command to execute the script.

The device schedules and initiates script execution.

Example:

```
<run/>
```

The system displays an RPC response confirming the script has been scheduled. For example:

```
<script-run-response>
          <response>Script run scheduled</response>
          <request-id>1622058854</request-id>
          <status>True</status>
          </script-run-response>
```

The script runs successfully. Monitor log files and system status as needed to verify completion and review outputs.

What to do next

After running the script, review its output, address any errors, and archive logs for future reference if necessary.

Exec scripts

An exec script is a network automation feature that

- enables users to execute predefined actions or sequences on network devices,
- makes use of remote procedure calls (RPCs) to interface with device data models, and
- supports integration with configuration, action, and operational data models.

Supported use cases for exec scripts

Exec scripts allow automation for common operational tasks, such as configuration backup, data collection, or network health monitoring.

Example of an exec script

For instance, an exec script can be triggered to periodically back up the device running configuration by calling an RPC on the `Cisco-IOS-XR-infra-script-mgmt-act` data model.

Add an exec script

Use this task to upload an exec script from an external source to the router's `harddisk:/mirror/script-mgmt/exec` script management repository for execution via script management features.

This task is used when you need to add a new executable script to automate operations or perform custom actions on a Cisco IOS XR device. You can use either the YANG data model interface or equivalent CLI commands based on your operational preference.

The **script add exec script-location script.py** is the equivalent command for `Cisco-IOS-XR-infra-script-mgmt-act.yang`.

Before you begin

Before you begin, ensure:

- You have the required exec script (for example, `sample1.py`) available on your file system or accessible from your management device.
- You have NETCONF/YANG or CLI access with sufficient privileges to execute script management operations.

Procedure

Step 1 Use the YANG data model to add the exec script via NETCONF RPC.

Example:

```
<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>
```

Upon completion, the system logs a message indicating that the script has been successfully added.

```
Router: script_manager[66762]: %OS-SCRIPT_MGMT-6-INFO :
           Script-script_manager: sample1.py has been added to the script repository
```

Step 2 Alternatively, use the CLI to add the exec script directly.

Example:

```
 script add exec harddisk:/sample1.py
```

The system confirms successful upload in the CLI output.

The exec script is now present in the script repository (harddisk:/mirror/script-mgmt/exec). It can be managed and executed as required.

What to do next

Optional: Verify the script is available by listing scripts in the repository using appropriate CLI command or NETCONF get operation.

- To list scripts via CLI:

  ```
  script list exec
  ```

Configure checksum

Ensure the integrity of script files by associating a checksum value, such as SHA-256, with the script.

Associating a checksum with a script protects against file tampering or corruption. You can configure the checksum using YANG data models or through the CLI, depending on your management preference.

Before you begin

Before you begin, ensure that:

- The script file you want to secure (.py, .sh, etc.) is present on the device.
- The calculated checksum value (for example, SHA-256 digest) is available.
- You have administrative access via CLI or NETCONF/YANG API.

Procedure

Step 1 Calculate the checksum value for your script file using a utility such as `sha256sum`.

Example:

```
sha256sum sample1.py
```

Step 2 Associate the checksum with the script using one of the following methods:

- CLI Method: Enter the following command from configuration mode::

```
script exec sample1.py checksum SHA256
5103a843032505decc37ff21089336e4bcc6a1061341056ca8add3ac5d6620ef
```

- NETCONF/YANG Method: Send an `edit-config` RPC to configure the checksum for the script file:

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

Step 3 If using NETCONF, verify that the response contains `<ok/>` to confirm successful configuration.

Example:

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

```
</rpc-reply>
```

Step 4 Verify that the checksum is applied by displaying the script properties using the appropriate CLI or YANG command.

Example:

```
Router# show script exec
```

The selected script now has an associated checksum value, protecting it against unauthorized modification or corruption.

What to do next

After configuring the checksum, test execution of the script to ensure proper operation and integrity verification.

Run an exec script

Use this task to execute a Python script (such as `sample1.py`) on the router for automation or system management functions.

You can run Python scripts from the router using either the CLI command or an RPC request through NETCONF/YANG interfaces. Successful execution provides operational insights or enables automation workflows.

Before you begin

- Ensure the Python script (`sample1.py`) is uploaded to the router's script directory.
- Verify you have sufficient privileges to execute scripts on the router.

Procedure

Step 1 Run the script using the CLI command.

Example:

```
script run sample1.py
```

A successful command schedules the script for execution and logs the activity in syslog.

The router executes the script and displays status output.

Step 2 Alternatively, run the script via NETCONF/YANG RPC request.

Example:

```
<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>
```

A successful execution returns an <ok/> response in NETCONF.

The script runs and the completion status is returned via RPC reply.

Step 3 Verify execution using syslog messages.

Example:

```
            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
```

Confirm that syslog entries show script scheduling and execution status for traceability.

The selected Python script executes successfully on the router. You can view output and confirmation in CLI, NETCONF response, or syslog entries.

What to do next

- Review script output and logs to ensure the script completed as intended.
- Address any errors or exceptions recorded in syslog.

Stop an exec script

Terminate a currently running exec script.

Use this task when you need to manually halt a script that is currently executing on the device. This action helps prevent unintended consequences if a script behaves unexpectedly or must be stopped for operational reasons.

Before you begin

Ensure you have administrative access to the device via CLI or NETCONF.

Procedure

Step 1 In the CLI, enter the following command to stop the exec script:

Example:

```
script stop <script-name>
```

Replace <*script-name*> with the name or identifier of the running script you want to stop.

Step 2 Alternatively, if you are using NETCONF, send an RPC request as shown:

Example:

```
<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>SCRIPT-ID</request>
          </script-stop-request>
          </rpc>
```

Replace *SCRIPT-ID* with the identifier for the script you want to stop.

The specified exec script is stopped.

What to do next

Verify the script has stopped by checking the script status. Restart or perform additional troubleshooting if required.

Remove an exec script

Remove an exec script from the script management repository to free up space or eliminate scripts that are no longer needed. This does not delete the script's management or execution history data.

Use this task when you want to remove a specific exec script from the repository. The removed script cannot be run unless you add it back again. The equivalent CLI for `Cisco-IOS-XR-infra-script-mgmt-act.yang` is **script remove exec <script_name.py>**

Before you begin

Ensure you have administrative access to the device or system where the script repository is managed.

Procedure

Step 1 Access the script management tool or command-line interface.

Step 2 List the available exec scripts to identify the script you want to remove.

Step 3 Remove the target exec script using the appropriate command or menu option.

For command-line removal, use:**script remove exec <script_name.py>**

Step 4 Verify that the script has been removed from the repository.

Example:

```
<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>
```

The selected exec script is removed from the repository and cannot be executed unless re-added.

What to do next

If needed, add a different script to the repository or clean up additional scripts as required.

View the script execution status

Check the progress and results of script execution on the device to confirm operational status or diagnose issues.

Network administrators may need to verify the status, logs, and outcomes of scripts that run on the device for troubleshooting or operational monitoring.

You can use either the CLI command or NETCONF RPC calls to view script execution status.

Before you begin

Ensure you have the required access privileges to run operational commands on the device (CLI or NETCONF), and that at least one script has been executed.

Procedure

Step 1 Use the CLI command to display script execution status and details.

Example:

```
show script execution [request-id <value>] [name <filename>] [status {Exception | Executed | Killed
 | Started | Stopped | Timed-out}] [reverse] [last <number>]
```

Replace options in brackets as needed to filter specific execution results by request ID, script name, or status.

The command displays script execution entries, including request ID, script name, current status, execution duration, and log path.

Step 2 Alternatively, send a NETCONF RPC request to retrieve script execution details programmatically.

Example:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="unique-id">
          <get>
          <filter>
          <script xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-script-mgmt-oper">
          <execution>
          <requests>
          <request>
          <request-id><your-request-id></request-id>
          <detail>
          <execution-detail/>
          </detail>
          </request>
          </requests>
          </execution>
          </script>
          </filter>
          </get>
          </rpc>
```

Replace <*your-request-id*> with the actual ID of the script execution you want to query.

A successful response returns execution summary, duration, log path, run options, and event history for the specified script execution.

Step 3 If more detail is needed, review the log path shown in the output.

Use the displayed log file path to access detailed execution logs for troubleshooting.

You have successfully retrieved the script execution status and details, which can now be used for monitoring or troubleshooting purposes.

What to do next

If issues are detected, use the log files or event details to further diagnose and resolve script execution problems.

EEM scripts using RPCs

EEM scripts using RPCs are automated network management tools that

- leverage Remote Procedure Calls (RPCs) to interact with device configuration and status
- support dynamic, event-based automation through the Cisco IOS XR event manager framework, and
- integrate programmable operations with data models such as Cisco-IOS-XR-um-event-manager-policy-map-cfg.yang and Cisco-IOS-XR-um-event-manager-cfg.yang.

Relevant data models and protocols

The following YANG data models support EEM scripts with RPC functionality:

- Cisco-IOS-XR-um-event-manager-policy-map-cfg.yang
- Cisco-IOS-XR-um-event-manager-cfg.yang (augmented for advanced programmability)

Using NETCONF or gNMI protocols, EEM scripts can launch device configuration changes or collect telemetry in response to specific network events.

Example of EEM script using RPCs

A sample EEM script is configured to monitor interface status. When an interface goes down, the script automatically triggers an RPC call to reset the interface and log the event for future auditing.

Configure event actions using the data model

Configure actions that automatically respond to specific network events by defining triggers and corresponding scripts using the YANG data model and NETCONF RPC.

Use this process to automate proactive handling of events (like syslog patterns) on NETCONF-enabled Cisco IOS XR devices. Event triggers and actions ensure operational responses are consistent and timely.

Before you begin

- Confirm NETCONF access and administrative credentials for your device.
- Ensure the device supports Cisco-IOS-XR-um-event-manager features.

- Prepare the event criteria (name, trigger pattern, severity) and action details (script, username, timeout, checksum).

Procedure

Step 1 Access the NETCONF configuration interface for your Cisco IOS XR device.

Step 2 Define an event trigger by setting these parameters.

a) Specify event-name (e.g., "event_1").
b) Set occurrence count and period in seconds.
c) Choose type as syslog, and provide the pattern and severity level.

Step 3 Configure an event action.

a) Enter action-name (e.g., "action_1").
b) Set type to script, with script-name, maxrun (timeout), and checksum values.
c) Provide username for script execution.

Step 4 Send the configuration using a NETCONF *edit-config* RPC request.

Example:

```
<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>
```

Step 5 Commit your configuration using the NETCONF *commit* RPC request.

Example:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
     <commit/>
   </rpc>
```

Step 6 Verify a successful NETCONF response includes `<ok/>`.

Example:

```
<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>
```

Your event-driven actions are now active and scripts are launched automatically when a matching event occurs.

What to do next

- Test the configuration by producing a matching syslog event on the device.
- Review system logs to confirm script execution and troubleshoot if needed.

Create a policy map for events and actions with a data model

You use a data model to create a policy map that defines actions for specific network events.

This enables automation and centralized management of event policies on supported devices.

Policy maps provide a reusable model for associating events with actions. By using the YANG data model, you can configure these policies programmatically and ensure consistent behavior across devices.

Before you begin

Before you begin, ensure the following:

- You have administrator access to the device.
- NETCONF is enabled on the device.
- You have the syntax details for the required YANG module and CLI commands.

Procedure

Step 1 Use the YANG data model `Cisco-IOS-XR-um-event-manager-policy-map-cfg` or the CLI to define a new policy map for event management.

Example:

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

Alternatively, configure the equivalent policy map and event-action relationship using the CLI:

event manager policy-map *policy-name*

action *action-name*

trigger event *event-name*

Step 2 Commit the changes to apply the configuration.

Example:

```
 <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
 <commit/>
 </rpc>
```

Step 3 Verify successful configuration.

If the RPC reply contains an <ok/> message, the configuration is successful.

The device is now configured with a policy map that assigns actions to specific events. Event policies are active and managed according to your definitions in the data model or CLI.

What to do next

Review policy operation and monitor event actions to ensure expected behavior. Adjust policy maps or event triggers as needed based on operational requirements.

Operational model for EEM script

An operational model is a framework that

- describes how a system, process, or component behaves under various conditions,
- defines the roles, actions, and interactions among different parts or actors during operation, and
- enables prediction, analysis, or management of real-world operational scenarios.

Table 7: 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.

Retrieve actions using the operational data model

The purpose of this task is to allow you to retrieve detailed information about specific IOS XR actions configured on the router by querying the operational data model with NETCONF. Action details are useful for validating scripts, troubleshooting, and monitoring operational status.

Actions in IOS XR are RPC statements that trigger operations or execute scripts. By querying the operational YANG model, you can programmatically view action attributes, recent status, run counts, and associations with policy maps.

This task is typically performed by network administrators or automation engineers managing IOS XR devices in programmable environments.

Before you begin

Before you begin, ensure these prerequisites are met:

- NETCONF is enabled and accessible on the IOS XR router.
- You have valid credentials to access the router via NETCONF.

- The target action name (for example, **action2**) is known.

Procedure

Step 1 Prepare a NETCONF RPC <get> request targeting the YANG module **Cisco-IOS-XR-ha-eem-policy-oper** and specify the action name you want to retrieve.

Example:

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

Replace **action2** with the actual action name you want to query.

Step 2 Send the NETCONF RPC request to your IOS XR device using your preferred NETCONF client or automation tool (such as ncclient or YANG Suite).

If using command-line tools, ensure the session is authenticated and the device supports the required YANG modules.

Step 3 Review the RPC response details to verify the action attributes and operational status.

Example:

```
            <rpc-reply message-id="urn:uuid:example-message-id"
xmlns="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>
```

Verify fields such as **script-name, last-run-status, triggered-count,** and **policy associations** for troubleshooting or validation purposes.

You have successfully retrieved the action details from the operational data model.

The system returns the operational attributes of the specified action in the NETCONF RPC response. This data allows you to automate monitoring, troubleshooting, or auditing of IOS XR actions and associated scripts.

What to do next

Optionally, repeat the query for other action names or use the retrieved output to update your automation records or configuration monitoring dashboards.

Retrieve a policy map using the operational data model

Retrieve the configuration and operational status of a policy map by submitting a NETCONF RPC request based on the Cisco-IOS-XR-ha-eem-policy-oper YANG model.

Use this task when you need to audit, troubleshoot, or verify the state and details of policy maps on Cisco IOS XR devices programmatically. This method provides a structured XML response containing policy actions, status, and event triggers.

Before you begin

- Ensure NETCONF is enabled on the target device.
- Have appropriate user credentials with permission to access operational data.
- Know the policy map name you want to retrieve.

Procedure

Step 1 Connect to the device via a NETCONF-capable client.
Connection to the device is established.

Step 2 Submit an RPC request using the Cisco-IOS-XR-ha-eem-policy-oper YANG model with the specific policy map name in your filter.

Example:

```
            <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>
```

The device returns a detailed XML response containing the policy map's configuration and operational status.

Step 3 Review the RPC response and verify details for the specified policy map, such as status, event map, action map, and occurrence counters.

Example:

```
            <rpc-reply message-id="urn:uuid:3cec3f3a-395b-4763-b1a1-1053149da60c"
xmlns="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-status>active</policy-status>
            <policy-occurrence>2</policy-occurrence>
            <event-count>2</event-count>
            <policy-event-map>
            <event-name>event5</event-name>
            <event-status>active</event-status>
            </policy-event-map>
            <policy-action-map>
            <action-name>action4</action-name>
            <checksum-enabled>SHA256</checksum-enabled>
            </policy-action-map>
            </policy-map-name>
            </policy-map-names>
            </eem>
            </data>
            </rpc-reply>
```

The requested policy map details are available and can be used for verification or troubleshooting purposes.

The specified policy map's operational details are successfully retrieved and displayed in XML format.

What to do next

- Optionally, store or export the policy map details for audit or ongoing monitoring.
- If further analysis is needed, reference the YANG model documentation for available attributes.

Retrieve events with trigger conditions using the operational data model

Retrieve and review event details by using the YANG operational data model to monitor event manager trigger conditions.

Use this procedure when you need to verify or troubleshoot event triggers registered via the event manager subsystem. This enables programmatic monitoring without relying solely on CLI output.

Before you begin

- Ensure you have NETCONF or RESTCONF access enabled on the target Cisco IOS XR device.
- Obtain device credentials with appropriate read privileges.
- Have the desired event trigger name(s) ready.

Procedure

Step 1 Establish a NETCONF session with the device using your preferred YANG management tool.

Step 2 Send a get operation with a filter that targets the event trigger data for the specific event name.

Example filter (replace *event4* with your event name):

Example:

```
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
          <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>
          </nc:get>
          </rpc>
```

Step 3 Review the returned RPC response for event status, type, details, and policy mapping.

Key fields: **event-status, event-type, event-policy-map, event-triggered-count.**

Example:

```
<event-name>event4</event-name>
          <event-status>active</event-status>
          <event-type>syslog</event-type>
           <event-policy-map>
           <policy-name>policy4</policy-name>
           </event-policy-map>
```

Step 4 Optionally, use the CLI for equivalent information if needed.

CLI command:

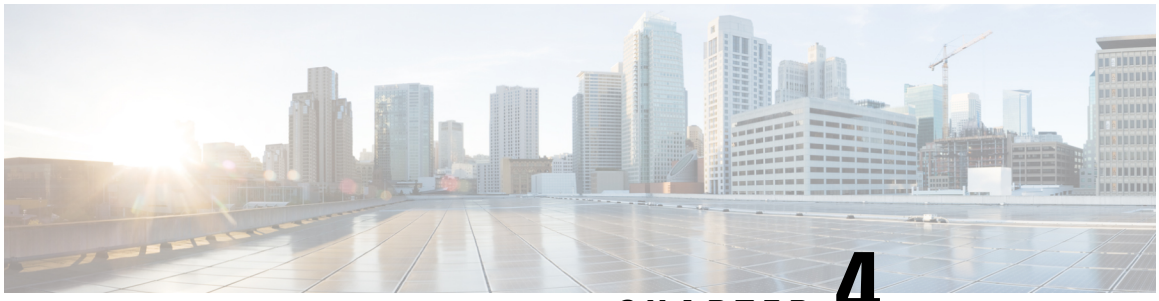
```
Router# show event manager event-trigger event-trigger-name detailed
```

For more details, refer to the View Operational Status of Event Scripts topic.

You have retrieved and verified the operational status and configuration of event triggers using the YANG data model or CLI.

What to do next

- Use the collected data for further troubleshooting, policy adjustment, or monitoring automation as needed.



CHAPTER 4

Precommit Scripts

Precommit scripts

A precommit script is a configuration management tool that

- executes custom python logic automatically during configuration commit operations,
- validates proposed configuration changes against administrator-defined policies, and
- determines whether to allow or block the commit based on script evaluation results.

Table 8: 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. These scripts are executed automatically during a configuration commit operation. They process the configuration change and act as a 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 play a crucial role in validating configurations during commit operations. These scripts are invoked automatically when a configuration change is committed. Device administrators use them to enforce custom validation rules and simplify repetitive configuration tasks.

During a configuration commit, precommit scripts are automatically initiated to validate changes. If the changes are valid, the script permits the commit. If not, it alerts the administrator to the mismatch and blocks the commit, maintaining device parameters and reducing human error.

Functions of precommit script

Precommit scripts are automatically invoked during a commit operation and can perform several functions.

- The system automatically invokes precommit scripts during a commit operation to validate configuration changes.
- The precommit scripts validate your proposed configuration to ensure it meets the required standards.
- These scripts ensure that any changes to the target configuration stay within system or software boundaries.
- The scripts can estimate the number of Ternary Content Addressable Memory (TCAM) slots that your configuration will require.
- The scripts verify that TCAM usage does not exceed the defined threshold.
- The scripts check that your commit operation follows the required execution rules, such as allowing configuration changes that affect traffic only during specified time intervals.
- If your configuration is invalid, the scripts block the commit operation and display a detailed error message.
- The scripts generate system log messages to help you analyze and troubleshoot failed commit operations.

Restrictions of precommit script

These 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.

Run the precommit script

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

Figure 1: Workflow to run precommit scripts

Before you begin

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.

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.

Procedure

Step 1 Download the script to the router.

Step 2 Configure checksum for precommit script.

Step 3 Activate precommit script.

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

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.

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.

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

Procedure

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

- Add the script from a server.
- Copy the script from an external repository.

a) Add the script from a configured remote server (HTTP, HTTPS, FTP or SCP) or the harddisk location in the router.

Example:

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

You can add a maximum of 10 scripts simultaneously. 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.

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.

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

Example:

```
Router#scp userx@192.0.2.0:/scripts/precommit-bgp.py /harddisk:/
```

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

Example:

```
Router#script add precommit /harddisk:/ precommit-bgp.py
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
=============================================================================================================

 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 the precommit script

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. 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.

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

Note Precommit scripts support SHA256 checksum.

Procedure

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
=============================================================================================================

 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
```

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.

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
=============================================================================================================

 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
2. Configure Checksum for Precommit Script

Procedure

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
Precommit Script Report Start
------------------------------
Pre-commit Verification Result: Pass
Pre-commit Verification Script precommit-bgp.py (req id 1656378102): Pass
------------------------------
Precommit Script Report Done
```

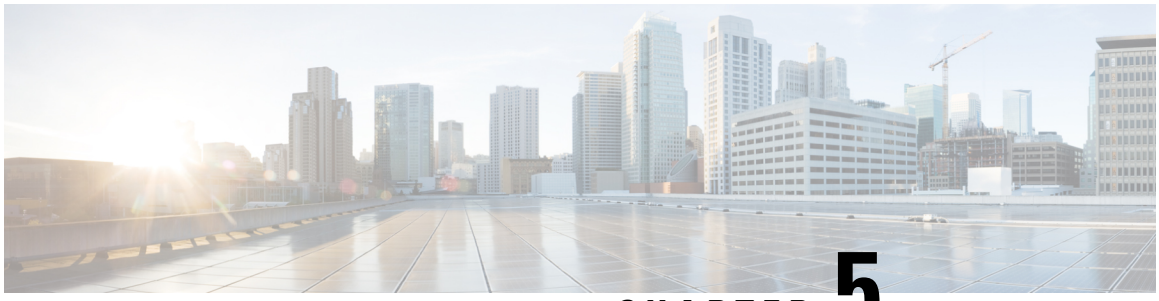
- Ignore the result of the precommit script execution and proceed to the next step in the commit process using **ignore-results** keyword. Use this keyword if you want to bypass the commit verification. The precommit script is still executed, but the result is ignored.

```
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.



CHAPTER 5

Config scripts

Config scripts

A config script is an automated script that

- validates, enforces rules, and can modify device configuration changes during the commit process,
- analyzes proposed configuration changes, blocks invalid configurations, or issues warnings as needed, and
- can automatically adjust or supplement configuration items to ensure compliance and reduce repetitive manual tasks.

Config scripts generate system log messages that aid in auditing and troubleshooting configuration changes.

Functions of config scripts

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

Restrictions for config scripts

Configuration and software restrictions when using config scripts include:

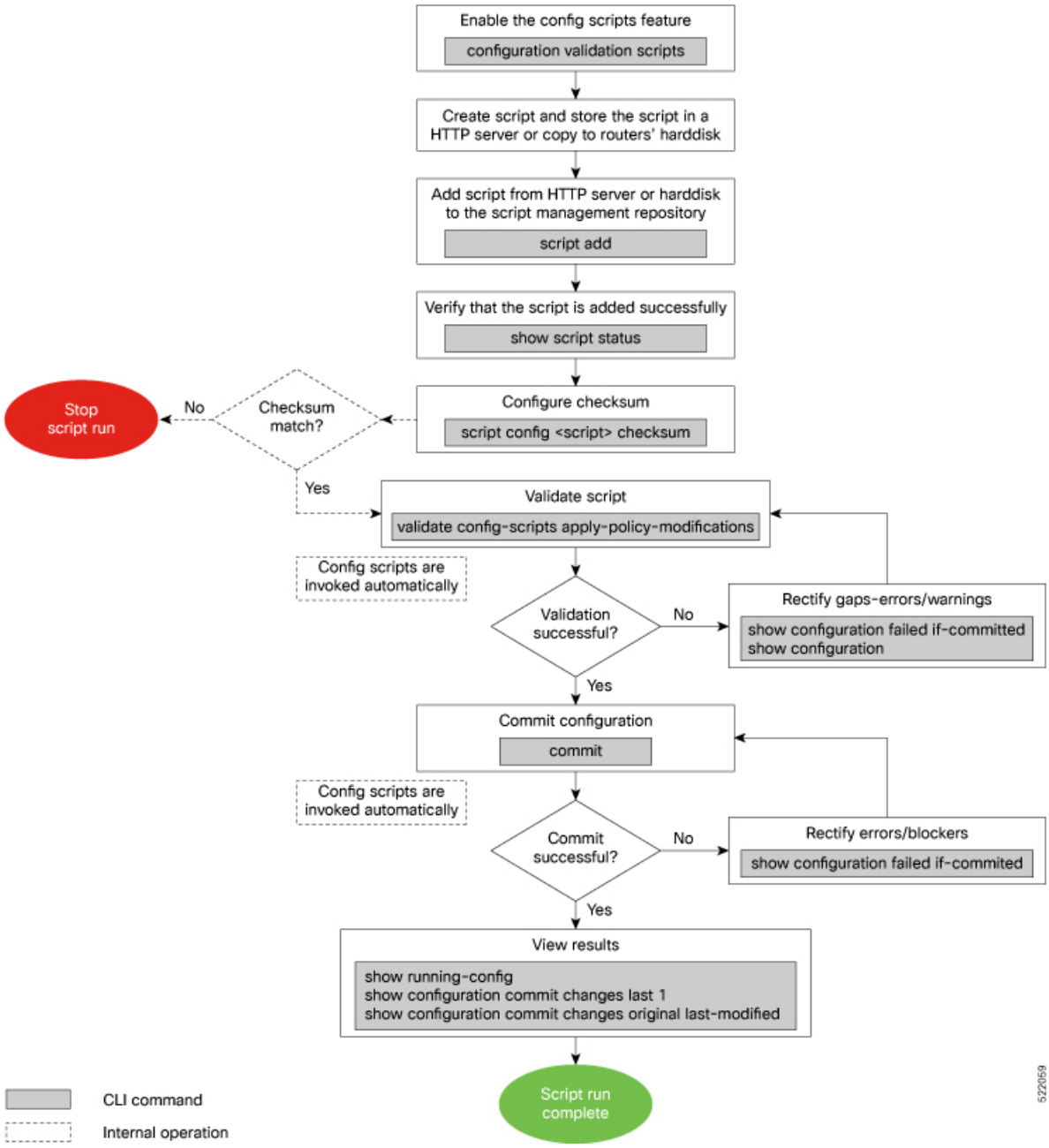
- Config scripts cannot modify configuration protected by the CCV process. This includes script checksum configuration and other sensitive security settings, such as AAA configuration.
- Config scripts do not support importing helper modules or custom imports for shared functionality. While you can configure these imports, they may introduce a security risk because there is no checksum validation on imported modules. Changes to these imported modules are not automatically detected. To have scripts recognize changes, you must manually unconfigure and reconfigure the affected scripts.

Run config scripts

Automate router configuration validation by enabling and executing configuration scripts.

Use this task to provision, validate, and apply Python-based configuration scripts on a router to ensure configuration changes meet organizational and operational standards.

Figure 2: Workflow to run config scripts:



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

Before you begin

- Ensure your configuration script is written in Python 3.5 using only Cisco-supported packages.

- Store the script on an HTTP server accessible to the router or on the router's local hard disk.
- Ensure you have privileges to run configuration and validation commands on the router.

Procedure

Step 1 Enable the config scripts feature.
Step 2 Download the script.
Step 3 Configure the checksum.
Step 4 Validate the script.
Step 5 Validate the configuration.
Step 6 View the script execution details.

Enable config scripts

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.

Procedure

Step 1 Enable the config scripts.

Example:

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

Step 2 Commit the configuration.

Example:

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

Download 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.

Table 9: Script download locations

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

Procedure

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

- Add script from a server.
- Copy the script from an external repository.

Example:

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
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.

Example:

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
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
================================================================================================================

 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
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 to the router

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.

Procedure

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
=============================================================================================================

 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
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.

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
=============================================================================================================

 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.

Configuration change validation

A configuration change validation is a network integrity mechanism that

- ensures that changes comply with predefined conditions set in active configuration scripts,
- evaluates both newly activated scripts and those previously active to catch any new or cumulative errors, and
- helps network administrators refine and update the target configuration before committing changes.

Pre-configuration validation

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.

Table 10: Feature History Table

Feature Name	Release Information	Description
Validate Pre-configuration Using Config Scripts	Release 25.1.1	Introduced in this release on: 8700 [ASIC: K100](select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.
Validate Pre-configuration Using Config Scripts	Release 24.1.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100(select variants only*)Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM
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.

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

Validate or commit configuration to invoke config script

Ensure that configuration changes meet policy conditions and are correctly applied using a config script before committing them.

Use this task when you want to validate and commit router configuration changes using a config script on IOS XR, ensuring modifications are applied as per policy.

Before you begin

- Confirm that your config script is available and properly installed on your device.
- Ensure you know the script's filename and checksum value if required.

Procedure

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

Example:

```
Router(config)#validate config-scripts apply-policy-modifications

% Policy modifications were made to target configuration, please issue 'show configuration'
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
```

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.

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

Example:

```
Router(config)#show logging
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.
```

Delete config script from the router

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

Before you begin

Verify that you know the name of the script you intend to delete.

Procedure

Step 1 View the active scripts on the router.

Example:

```
Router#show script status
=================================================================================================================

 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
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
### 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
```

Set script execution priority

Ensure multiple scripts modifying the same configuration items are executed in the required order.

When more than one script can change the same configuration item, execution order determines which script's change is applied last. Using priorities helps prevent unwanted dependencies and ensures correct validation.

Before you begin

Ensure all scripts are loaded and accessible on the router.

Procedure

Step 1 Assign a lower numerical priority value to scripts that should run first.

Example:

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

Step 2 Assign a higher numerical priority value to scripts that should run later.

Example:

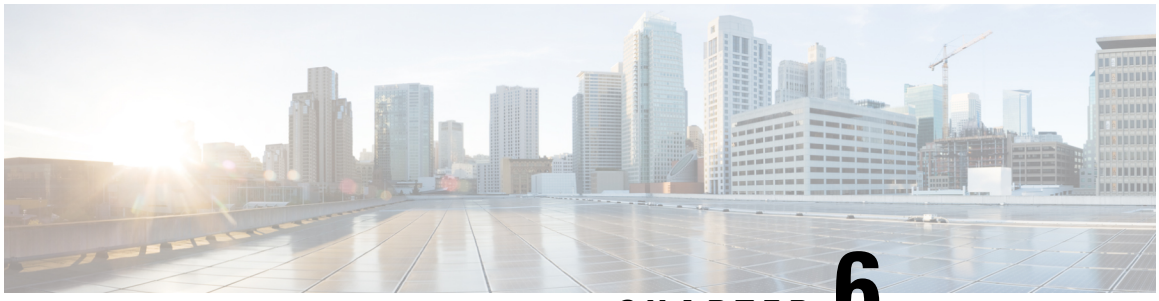
```
Router(config)#script config sample2.py checksum sha256
2fa34b64542f005ed58dcaa1f3560e92a03855223e130535978f8c35bc21290c
priority 20
```

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`.



CHAPTER 6

Exec Scripts

Exec scripts

Exec scripts are on-box scripts that

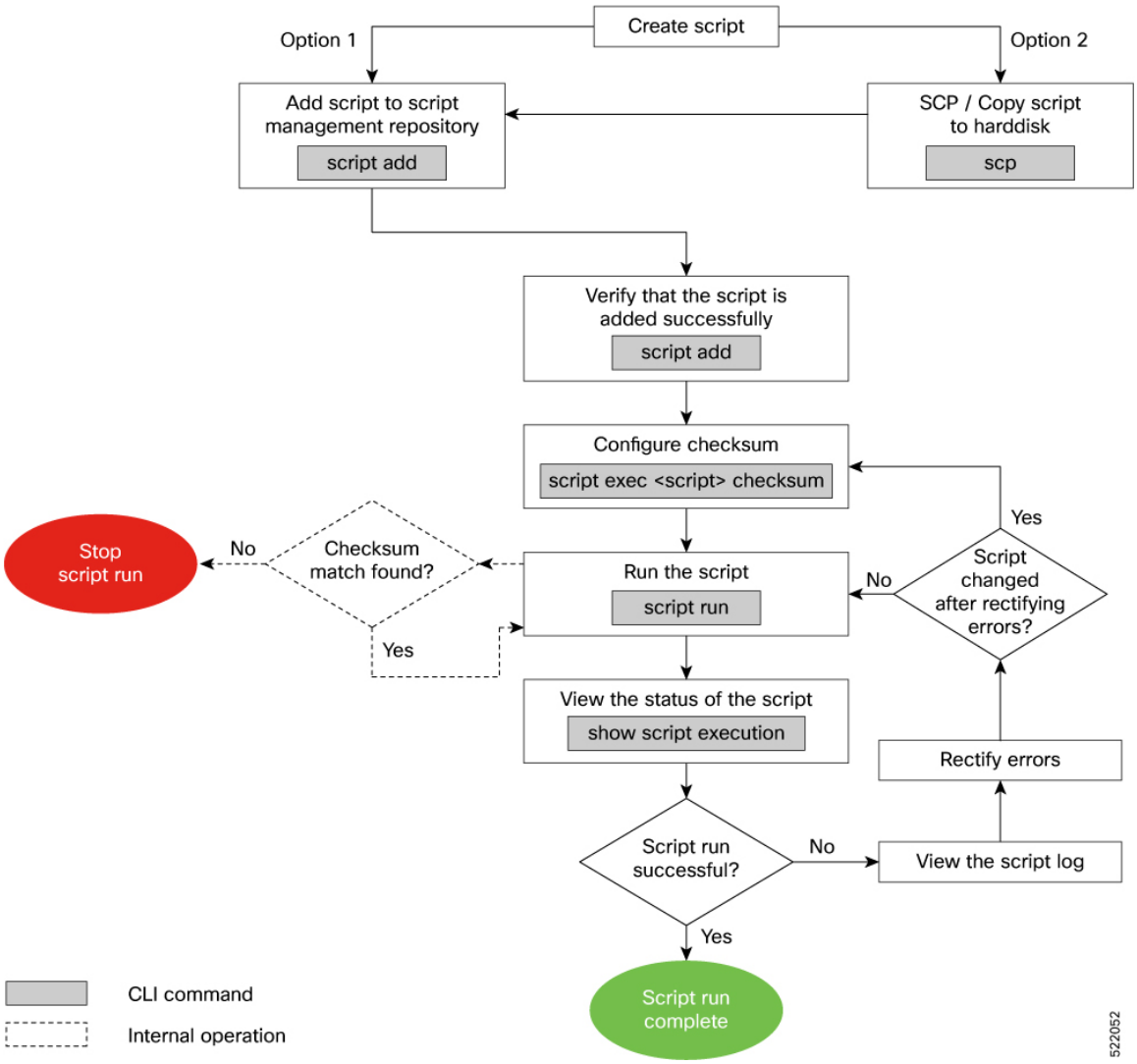
- automate device configurations throughout the network,
- are written in Python using libraries provided by Cisco with the base package, and
- execute within the Cisco IOS XR operating system.
- A script management repository on the router manages exec scripts and is replicated on both route processors (RPs).
- In IOS XR, AAA authorization determines user permissions required to run exec scripts; root privileges are necessary.

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.

Provision an exec script

Set up and execute an exec script on a router to automate administrative or configuration tasks.

Figure 3: Steps involved in provisioning an exec script.



Procedure

Step 1 Download the script.

Step 2 Configure checksum.

Step 3 Run the script.

Step 4 View the script execution details.

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.

Table 11: Script download locations

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

Procedure

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

- Add script from a server.

```
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.

```
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.

Example:

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

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

Example:

```
Router#script add exec /harddisk:/ exec-script.py
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
=============================================================================================
 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

A remote script update capability is a script management feature that

- enables routers to synchronize automation scripts from a master script stored at a remote repository,
- automates the update process by applying changes based on predefined triggers such as manual requests, scheduled intervals, or execution events, and
- supports multiple transfer protocols and configurable authentication to ensure reliable and secure distribution of script updates.

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.

Table 12: Feature History Table

Feature Name	Release information	Description
Update Automation Scripts from Remote Server	Release 25.1.1	Introduced in this release on: 8700 [ASIC: K100](select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.

Feature Name	Release information	Description
Update Automation Scripts from Remote Server	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100(select variants only*)Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM
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.

Update scripts from a remote server

You can update script from a remote server using one of the following options:

- Config CLI commands
- Exec CLI commands

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.

Procedure

Step 1 Update the script on the router with the version on the remote server.

Example:

```
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.

Table 13: Script update methods and options

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.

Condition	Description
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.

Step 2 Commit the configuration.

Example:

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

Step 3 Run the script.

Example:

```
Router#script run sample3.py background
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.

Step 4 Update the script on the router with the version on the remote server.

Example:

```
Router#script update manual exec sample2.py
sample2.py has been added to the script repository
```

You can set options when updating the script:

Table 14: Description of various keywords

Option	Description
WORD	Script name.

Option	Description
all	Update all scripts in config.

Invoke scripts from a remote server

Execute a script remotely using a specified protocol URL and checksum for verification.

Use a remote server URL to run Python scripts. The checksum is required to ensure script integrity. Supported protocols include FTP, HTTP, HTTPS, TFTP, and SCP. Additional command options are available:

- arguments: Script command-line arguments, as strings in single quotes.
- description: Description of the script run.
- log-level: Script logging level (default: INFO).
- log-path: Location to store script logs.
- max-runtime: Maximum runtime for the script.
- vrf: Specify the VRF for the network file system.

Procedure

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
```

Configure the checksum for an exec script

Ensure the integrity of exec scripts by configuring and verifying a SHA256 checksum, preventing tampering.

Every exec script requires a configured checksum. SHA256 is supported. The router verifies the checksum before executing the script. If the values do not match, the script will not run and a syslog warning is displayed.

Before you begin

- Ensure that the script is added to the script management repository. See Download the Script to the Router.
- Retrieve the SHA256 checksum value for your script on a trusted device.

Procedure

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:

```
Router#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
=============================================================================================================

 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
Router(config)#end
```

Step 5 Verify the status of the script.

Example:

```
Router#show script status detail
=============================================================================================================

 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 an exec script

Execute an exec script on the router using the script run command and receive a unique request ID for tracking the execution.

Scripts allow automation of functions on your router. You can specify various runtime options such as arguments, descriptions, logging level, and maximum runtime when running a 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.

Scripts can be run with more options. This table lists the various options that you can provide at run time:

Table 15: Runtime configuration options

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.

Before you begin

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

1. Download the Script to the Router
2. Configure Checksum for Exec Script

Procedure

Run the exec script.

Example:

```
Router#script run sample1.py
Script run scheduled: sample1.py. Request ID: 1629800603
Script sample1.py (exec) Execution complete: (Req. ID 1629800603) : Return Value: 0 (Executed)
```

What to do next

Review the script execution results and logs as needed. Use the request ID to track or troubleshoot any issues with the script run.

View the script execution details

View the status of the script execution.

Before you begin

Ensure you have completed these tasks.

- Download the script to the router.
- Configure checksum for exec script.
- Run the exec script.

Procedure

Step 1 View the status of the script execution.

Example:

```
Router#show script execution
==========================================================================================================

 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
==========================================================================================================

 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      :
==========================================================================================
```

Table 16: Description of various keywords

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`
status	Filter data based on script status. `Router#[status {Exception,Executed, Killed, Started, Stopped, Timed-out}]`

Delete exec scripts from the router

Remove exec scripts from the router's script management repository.

Use this task to delete unwanted exec scripts stored in the script management repository using the `script remove` command. The repository stores downloaded scripts available for use on the router.

Procedure

Step 1 View the list of scripts present in the script management repository.

Example:

```
Router#show script status
=============================================================================================================
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
sample2.py has been deleted from the script repository
```

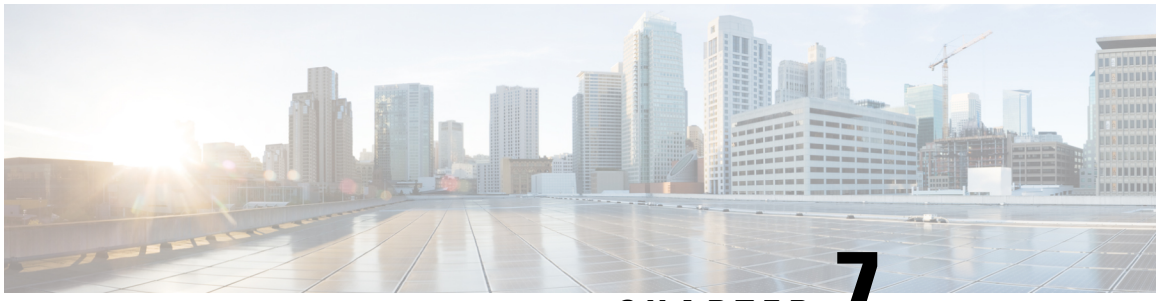
You can also delete multiple scripts simulataneoulsy.

Step 3 Verify the script is deleted from the subdirectory.

Example:

```
Router#show script status
=============================================================================================================
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.



CHAPTER 7

Process Script

Process scripts

A process script is a persistent automation script that

- runs continuously on Cisco IOS XR as long as it is activated,
- is managed by the AppMgr, which controls startup, monitoring, and restart, and
- registers as an application instance for operational monitoring and status retrieval.

Process scripts, also referred to as daemon scripts, are designed to execute on Cisco IOS XR platforms without manual intervention. The AppMgr is responsible for starting, stopping, monitoring, and retrieving the status of these scripts. AppMgr ensures that scripts automatically restart if they terminate depending on the configured restart policy and manages startup dependencies with other processes on the router.

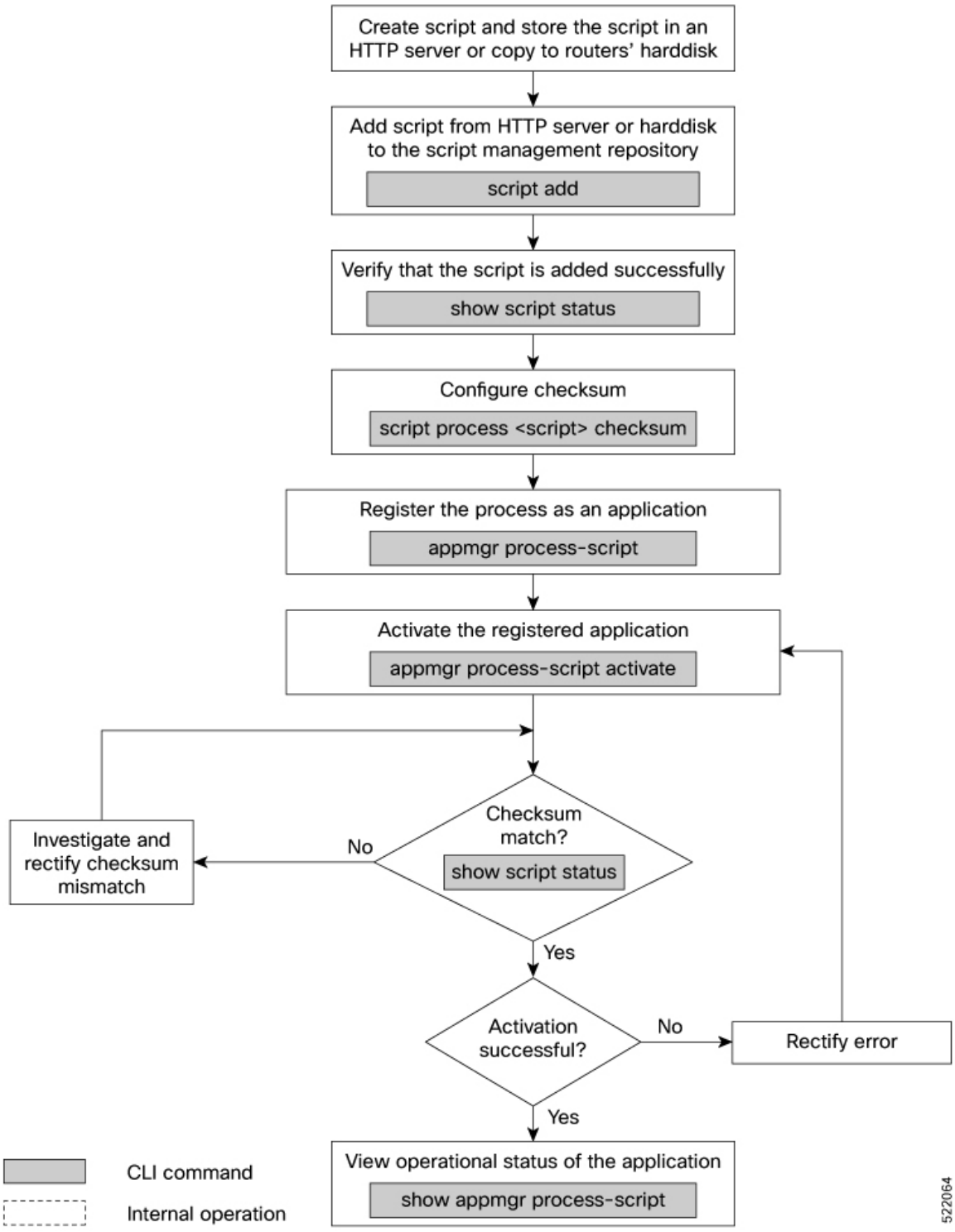
Process scripts support Python 3.5 as their programming language. For information on supported Python packages, refer to Cisco IOS XR Python Packages.

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

Run the process script

Run a process script to automate a specific set of system tasks.

Figure 4: Workflow of Process script



Before you begin

Procedure

Step 1 Download the script.
Step 2 Configure the checksum.
Step 3 Register the script.
Step 4 Activate the script.
Step 5 View the script execution details.

What to do next

Download the script to the router

Add a script to the script management repository on the router to enable script execution and management.

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.

Table 17: Location to download the script

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

Before you begin

- Confirm you have access to the script file(s) and to the router's CLI.
- If copying from an external repository, ensure required network connectivity and protocol support (HTTP, HTTPS, FTP, TFTP, SCP).

Procedure

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.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
  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 Repositor

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

  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:/
  ```

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

  ```
  Router#script add process /harddisk:/ process-script.py
  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
=============================================================================================
 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 the process script

Ensure script integrity by configuring a checksum that verifies the script has not been tampered with before execution.

Each process script is associated with a SHA256 checksum value, which acts as a fingerprint to ensure the script's integrity. If the configured checksum does not match the calculated value during script execution, the script will not run, and a warning is displayed. Configuring the checksum is mandatory for running the script.

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 for detailed instructions.

Procedure

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
=============================================================================================
 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
Router(config)#end
```

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
=============================================================================================
 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

Enable and manage a process script on your router by registering it as an application, ensuring it runs under the app manager and applies the desired execution settings.

Registering the process script with the app manager is a required step before you can use the script on the router. This process allows you to define application parameters and control restart policies for continuous operation or error handling.

Before you begin

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

- Download the Script to the Router
- Configure Checksum for Process Script

Procedure

Step 1 Register the script with an application (instance) name in the app manager.

Example:

```
Router#configure
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.

Table 18: 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
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
- Configure Checksum for Process Script
- Register the Process Script as an Application

Procedure

Step 1 Activate the process script.

Example:

```
Router#appmgr process-script activate name my-process-app
```

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
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 and review operational data and logs for a process script on the router.

Use this task when you need to monitor process script health, analyze performance, or troubleshoot errors on your router. This ensures accurate status information and supports problem resolution.

Before you begin

Ensure that the following prerequisites are met before you obtain the operational data:

- Download the Script to the Router
- Configure Checksum for Process Script
- Register the Process Script as an Application
- Activate the Process Script

Procedure

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
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.

The following example shows the output logs:

Example:

```
Router#show appmgr process-script my-process-app logs output
[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:

Example:

```
Router#show appmgr process-script my-process-app logs errors
----------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:

Example:

```
Router#show appmgr process-script my-process-app logs errors
----------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----------
```

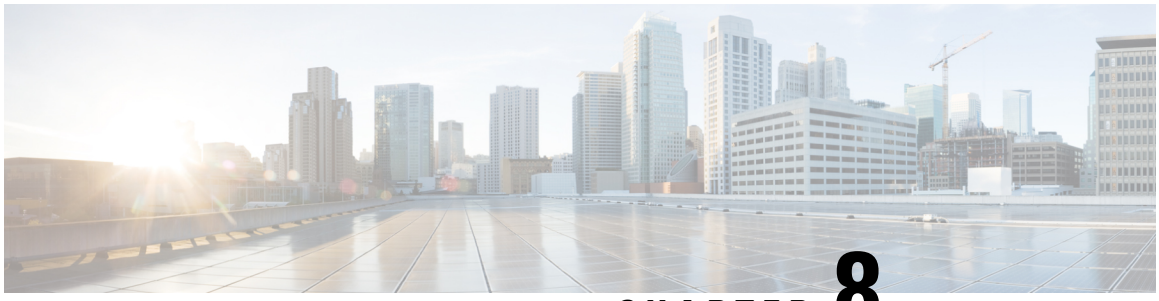
```
----------Run ID:1624346225  Fri Aug 20 11:08:23 2021----------
----------Run ID:1624346226  Fri Aug 20 11:09:29 2021----------
----------Run ID:1624346227  Fri Aug 20 11:10:35 2021----------
----------Run ID:1624346228  Fri Aug 20 11:11:41 2021----------
```

Manage actions on process script

The process script runs as a daemon continuously. You can, however, perform these actions on the process script and its application.

Table 19: Supported actions for process script applications

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`**



CHAPTER 8

EEM scripts

EEM scripts

A EEM script is an automation tool that

- monitors real-time system activity and events,
- executes defined actions when specific conditions are met, and
- streamlines troubleshooting and operational workflows.

EEM scripts act based on significant system occurrences—such as log messages, interface states, or telemetry changes—not limited to errors. For example, you can automate actions like enforcing LACP dampening if a bundle interface flaps multiple times within seconds by temporarily disabling the interface.

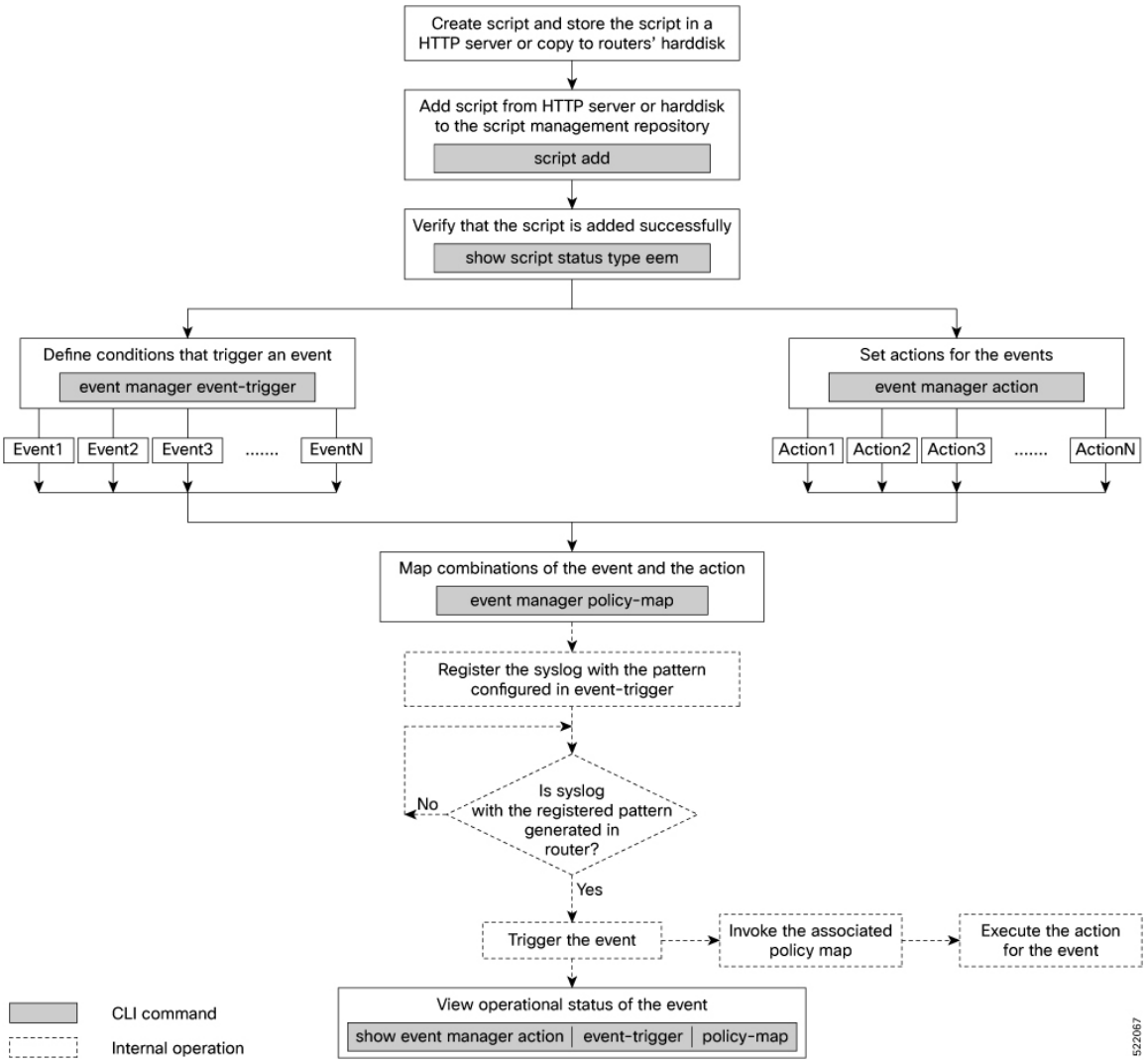
Administrators can define events and actions separately, then link them together using policy maps. A single event and action pair can be reused across multiple policies, and a policy map can contain up to five actions. EEM scripts may be written in Python 3.5 or TCL; script policies can be configured using either Command Line Interface (CLI) or NETCONF RPCs.

- Scripts are stored and managed in subdirectories based on type within the router's script management repository.
- You can map the same event and action in up to 64 policy maps for extensive automation reuse.
- For python scripting, only specific packages are supported; TCL scripts are also an option for event managers.

Manage eem scripts on the router

Complete these steps to provision eem scripts on the router.

Figure 5: Workflow to run events script



Procedure

Step 1 Download the script.

Step 2 Define events.

Step 3 Create actions to the events.

Step 4 Create policy map.

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.

Step 5 View operational status of the event.

Download 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.

Table 20: Script download locations

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

Procedure

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

- Add script from a server.
- Copy the script from an external repository.

Example:

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 config 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
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.

Example:

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
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
=============================================================================================
 Name               | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
 eem-script.py      | eem      | Config Checksum  | NEW         | Tue Aug 24 10:44:53 2021
=============================================================================================
```

Define trigger conditions for events

Configure system criteria so specific events are automatically triggered when defined conditions are met.

- The keywords `occurrence` (number of matches before event raises) and `period` (interval for above) can be used with syslog events only.
- To verify a telemetry sensor path or query before configuring an event trigger, use:

  ```
  Router# event manager telemetry sensor-path <sensor-path> json-query <query>
  ```

- Example for syslog trigger with severity:

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

  The event triggers if both pattern and severity match a syslog message.

Before you begin

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

Procedure

Step 1 Register the event.

Example:

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

Step 2 Configure the trigger type and its options for the event.

Syslog event

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

- Specify a pattern that matches the syslog message.
- Optionally, add a severity value (alert, critical, debug, emergency, error, info, notice, warning).
- The event triggers only when both pattern and severity match, or if severity is not set, any severity matches the pattern.

Timer event

Watchdog timer

```
Router(config)# event manager event-trigger <event-name> type timer watchdog value <seconds>
```

Cron timer

```
Router(config)# event manager event-trigger <event-name> type timer cron cron-entry "<cron string>
```

Track event

```
Router(config)# event manager event-trigger <event-name> type track name <track-name> status {up |
down | any}
```

Triggers when the specified track object's status changes.

Telemetry event:

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 <sensor-path> sample-interval <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 <sensor-path> sample-interval <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 <sensor-path>
sample-interval <seconds>
```

Before creating a telemetry event, enable model-driven telemetry:

```
Router# telemetry model-driven
```

The router or system will now automatically trigger the event when the specified conditions are matched.

Create actions for events

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

Before you begin

- Define trigger conditions for an event.

Procedure

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
```

If you load the event manager action commands using configuration files, for example, by using the **load harddisk:config.txt** command, you must make sure that the commands in the configuration files are properly indented and aligned with the running configuration.

In this example, the **username eem** and **type script** commands in the **config.txt** configuration file are properly indented and aligned with the running configuration.

```
event manager action action_all
 username eem
 type script script-name eem.py Marx seconds 7200 checksum
sha256fb2e1f7c4b135c296abb7149cf5fb96f052d3876c35a8422d44f78b9b6d3e452
```

Policy maps

Policy map is a configuration object that

- enables the association of multiple actions with one or more events,
- supports boolean logic (AND or OR) for correlating multiple events, and
- allows conditional triggering based on event occurrences within a specified period.

With a policy map, you can configure the system to execute an EEM (Embedded Event Manager) script only when certain combinations of events happen, such as requiring both a specific status threshold and a timer event before taking action. Optional parameters like occurrence and period control how frequently or under what time constraints the policy is triggered.

Table 21: Feature History Table

Feature Name	Release Information	Description
Add Multiple Events In a Policy Map With a Single EEM Script	Release 25.1.1	Introduced in this release on: 8700 [ASIC: K100](select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.
Add Multiple Events In a Policy Map With a Single EEM Script	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100(select variants only*)Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM

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.

Associate events and actions with a policy map

Configure a policy map to link network events with automated actions using Embedded Event Manager (EEM).

Policy maps allow you to define which events trigger specific actions on your router. You can use Boolean logic to correlate multiple events, specify how many occurrences trigger the action, and set a time period for event evaluation.

Before you begin

- Ensure you are in global configuration mode on the router.
- Identify the EEM events and actions you want to associate.

Procedure

Step 1 Start a policy map configuration.

Example:

```
Router(config)#event manager policy-map policy1
```

Step 2 Define event triggers.

For a single event

Example:

```
Router(config-policy-map)#trigger event event1
```

For multiple events with Boolean logic, enclose the logic in double quotes:

Example:

```
Router(config-policy-map)#trigger multi-event "<event1> AND (<event2> OR <event3>
```

Step 3 (Optional) Set occurrence count.

Specify how many times the total correlation must occur before the event is raised

Example:

```
Router(config-policy-map)# occurrence 2
```

- Acceptable values: 1 to 32.

- If not specified, the policy map triggers on every occurrence

Step 4 (Optional) Set evaluation period.

Define the time interval (in seconds) during which the events must occur.

```
Router(config-policy-map)# period 60
```

Acceptable values: 1 to 429496729.

Step 5 Specify actions.

Map actions to the policy map (maximum of 5).

```
Router(config-policy-map)# action action-2
```

The policy map is configured. When defined events occur according to your logic, the router automatically executes the associated action.

View operational status of eem components

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

Before you begin

- Define trigger conditions for an event
- Create actions for events
- Create a policy map of events and actions

Procedure

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

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

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
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

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

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.

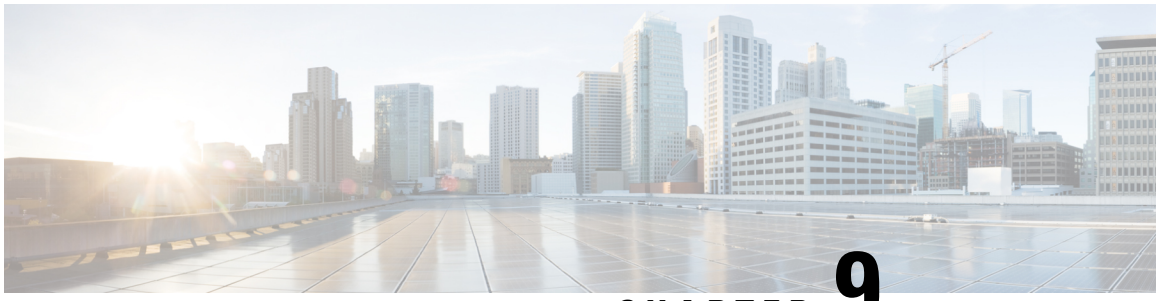
Example:

```
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.



CHAPTER 9

Model Driven Command Line Interface

Model-driven CLI features for data model visualization

A model-driven CLI feature is a programmability capability in Cisco IOS XR software that

- enables users to display YANG data model structures and operational data directly via CLI commands,
- provides configuration and operational output in both XML and JSON formats for easier parsing and automation, and
- introduces specialized CLI commands that help transition between traditional CLI and model-driven approaches.

Model-driven CLI features are designed to bridge the gap between legacy command-line workflows and modern, programmatic network management. By leveraging YANG-based models, these features allow network operators to view, query, and retrieve data in structured formats, streamlining the integration with external tools and automation systems.

Table 22: 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. Show commands present unstructured data, while data models provide structured data in XML or JSON formats; however, both methods can show different views. The model-driven CLI feature addresses

adoption challenges by allowing operators to use the familiar CLI while accessing structured, model-driven output. This enables easier integration with parsing scripts and automation tools.

Structure of data model

The structure of a data model organizes configuration and operational data in a hierarchical format. Each data model consists of these primary components:

- Model: The highest-level YANG file that defines a data schema e.g., `ietf-interfaces.yang`.
- Module: A logical collection of related definitions within the model e.g., `ietf-interfaces`.
- Container: A grouping node that organizes related nodes together e.g., `interfaces`, `interfaces-state`.
- List/Node: An array or list element within a container, often representing multiple items e.g., `interface* [name]`.
- Leaf: A single value node containing configuration or state data e.g., `name`, `description`, `type`, `enabled`, `admin-status`.

Table 23: Summary table of data model structure:

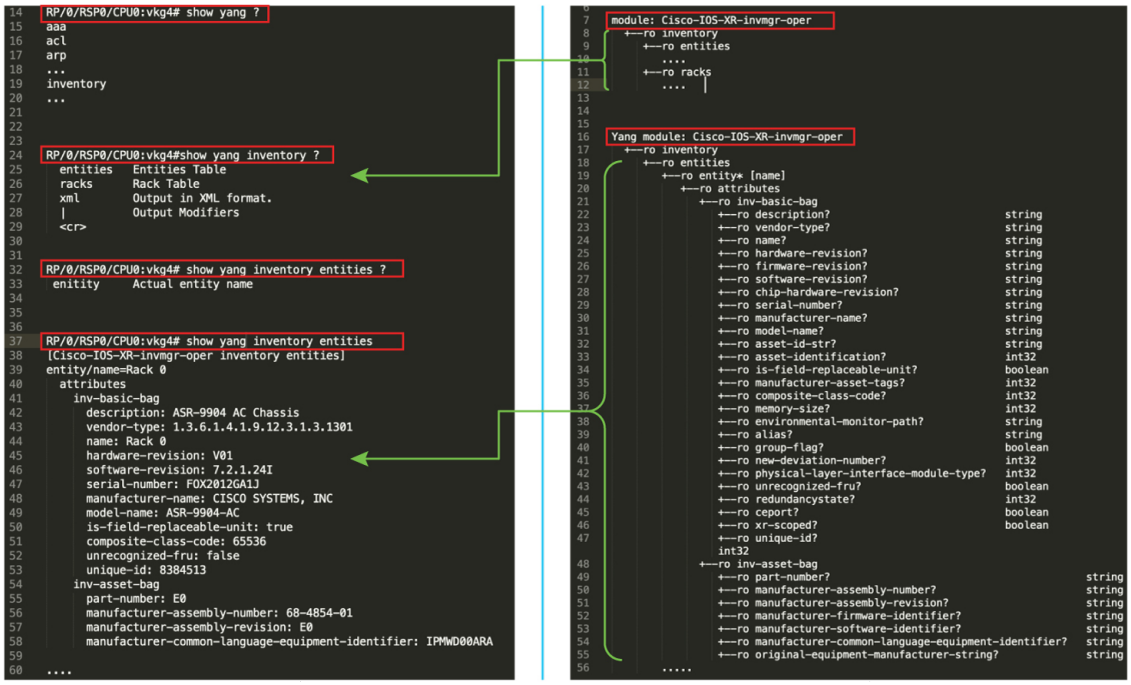
Component	Description	Example
Model	YANG file containing schema definitions	`ietf-interfaces.yang`
Module	Group of related schema nodes	`ietf-interfaces`
Container	Grouping node for related lists and leaves	`interfaces`, `interfaces-state`
List/Node	Multiple entries identified by unique key	`interface* [name]`
Leaf	Single data value node under a container or list	`name`, `type`, `enabled`, etc.

CLI and navigation notes:

- Use the **show yang operational** command to explore down to the leaf level in a data model, similar to navigating hierarchical data.
- Data model structure guides how CLI outputs and configurations map to YANG-defined nodes.

The image show a mapping between CLI and data model, and how the structured data is displayed on the console.

Figure 6: Mapping between CLI and Data model



Navigating YANG operational data models via CLI commands

YANG operational data models can be queried and explored using various CLI commands on Cisco network devices. These commands help you efficiently access operational state data, identify container and node structures, and retrieve detailed model attributes for troubleshooting and monitoring. Below are the main features, queries, outputs, and usage instructions for navigating YANG operational data.

Table 24: Key queries for YANG operational data

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`
	`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`

Operational Query	Description

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

A model-driven CLI display format is a configuration output method that

- presents device running configuration in structured XML and JSON formats,
- leverages native, OpenConfig, and unified YANG data models to organize configuration data, and
- enables granular filtering and flexible retrieval of configuration for specific components or hierarchies.

Table 25: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM
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 for the show run command

The **show run | xml** command produces the router's running configuration in XML format. This enables automation and integration with software that can parse device settings through a standard XML schema.

```
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 for show run command

he `show run | json` command on Cisco IOS XR displays the device's running configuration in JSON format, making it easier for automation tools, scripts, or APIs to parse and manipulate the configuration data programmatically.

A typical output includes:

- Interface configuration details, such as interface name, state, and protocol.
- Netconf/YANG agent configuration, such as SSH status.

```
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 for show run command

The output of the **show run router bgp <as-number> neighbor <neighbor-address> | json unified** command provides a structured, highly detailed view of BGP configuration for a specified neighbor in JSON format. This reference explains how to interpret each component of the output and highlights key attributes.

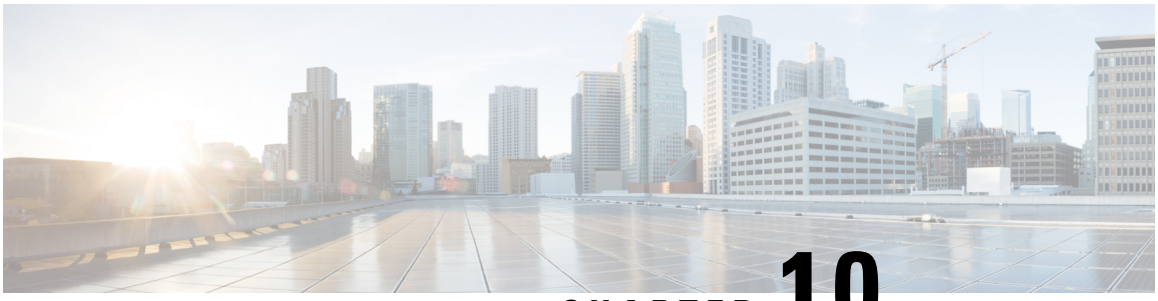
```
Router#show run router bgp 12 neighbor 12.12.12.12 | json unified
{
 "data": {
  "Cisco-IOS-XR-um-router-bgp-cfg:router": {
   "bgp": {
    "as": [
     {
      "as-number": 12,
      "neighbors": {
       "neighbor": [
        {
         "neighbor-address": "12.12.12.12",
         "remote-as": 12,
         "address-families": {
          "address-family": [
           {
            "af-name": "ipv4-unicast"
           }
          ]
         }
        }
       ]
      }
     }
    ]
   }
  }
 }
}
```

Unified model output for show run command

The following sample output demonstrates the unified model XML structure produced by the **show run router bgp 12 | xml unified** command:

```
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>
```

This output helps you identify the XML element hierarchy and data structure when configuring BGP settings using the unified model on Cisco IOS XR devices.



CHAPTER 10

Automation Scripts

Operational simplicity using automation scripts

An automation script is a software program that

- automates configuration and operational tasks on network devices,
- interacts with network operating systems through standard interfaces such as NETCONF, SNMP, and SSH, and
- can be executed either externally or directly on the network device.
- On-box automation scripts: Scripts that reside and execute directly on the router, eliminating the need for external controllers.
- Off-box automation scripts: Scripts that run on external controllers and interact with network devices through APIs over the network.

Table 26: Feature History Table

Feature Name	Release Information	Description
Operational Simplicity Using Automation Scripts	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.

Feature Name	Release Information	Description
Operational Simplicity Using Automation Scripts	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM
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: • Config scripts—Implement custom configuration rules, and notify the user to take action when the configuration conditions are not met. • Exec scripts—Automate operational tasks and network troubleshooting. • Process scripts—Monitor the system continuously using daemons. • EEM scripts—Respond to a predefined set of events.

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

This table provides the scope and benefit of on-box scripts.

Table 27: On-Box Automation Scripts

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