Programmability Configuration Guide for Cisco NCS 540 Series Routers, Cisco IOS XR Release 7.5.x

First Published: 2021-11-30

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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2021 Cisco Systems, Inc. All rights reserved.

CONTENTS

CHAPTER 1 **Drive Network Automation Using Programmable YANG Data Models** 1

- YANG Data Model 2
- Access the Data Models 9
- CLI to Yang Mapping Tool 10
- Communication Protocols 11
  - NETCONF Protocol 12
  - gRPC Protocol 12
- YANG Actions 12

CHAPTER 2 **Use NETCONF Protocol to Define Network Operations with Data Models** 17

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

CHAPTER 3 **Use gRPC Protocol to Define Network Operations with Data Models** 37

- gRPC Operations 40
- gRPC Network Management Interface 41
- gRPC Network Operations Interface 41
- Configure Interfaces Using Data Models in a gRPC Session 41

CHAPTER 4 **Unified Configuration Models** 49

PART I **Automation Scripts** 57

CHAPTER 5 **Achieve Network Operational Simplicity Using Automation Scripts** 59

Explore the Types of Automation Scripts 59

CHAPTER 6 **Config Scripts 61**

Workflow to Run Config Scripts 62

Enable Config Scripts Feature 63

Download the Script to the Router 64

Configure Checksum for Config Script 65

Validate or Commit Configuration to Invoke Config Script 67

Manage Scripts 69

Delete Config Script from the Router 69

Control Priority When Running Multiple Scripts 70

Example: Validate and Activate an SSH Config Script 71

Scenario 1: Validate the Script Without SSH Configuration 72

Scenario 2: Configure SSH and Validate the Script 73

Scenario 3: Set Rate-limit Value to Default Value in the Script 74

Scenario 4: Delete SSH Server Configuration 75

CHAPTER 7 **Exec Scripts 77**

Workflow to Run an Exec Script 77

Download the Script to the Router 79

Configure Checksum for Exec Script 80

Run the Exec Script 82

View the Script Execution Details 83

Manage Scripts 85

Delete Exec Script from the Router 85

Example: Exec Script to Verify Bundle Interfaces 86

CHAPTER 8 **Process Scripts 93**

Workflow to Run Process Scripts 93

Download the Script to the Router 96

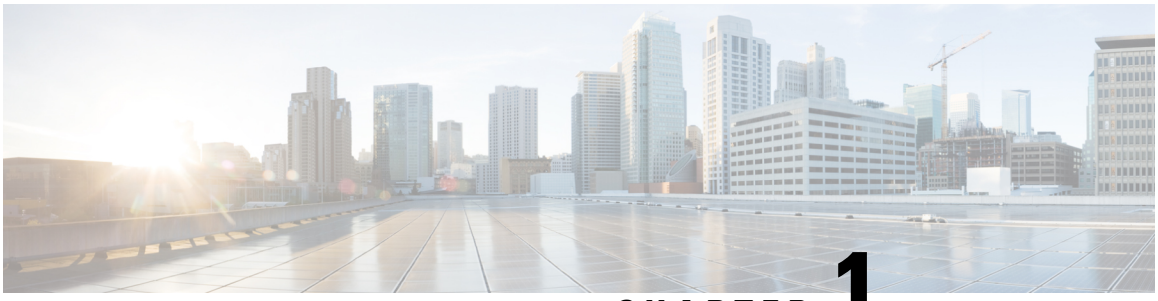
Configure Checksum for Process Script 97

Register the Process Script as an Application 99

Activate the Process Script 100

Obtain Operational Data and Logs 101

Managing Actions on Process Script 103
Example: Check CPU Utilization at Regular Intervals Using Process Script 103



CHAPTER 1

Drive Network Automation Using Programmable YANG Data Models

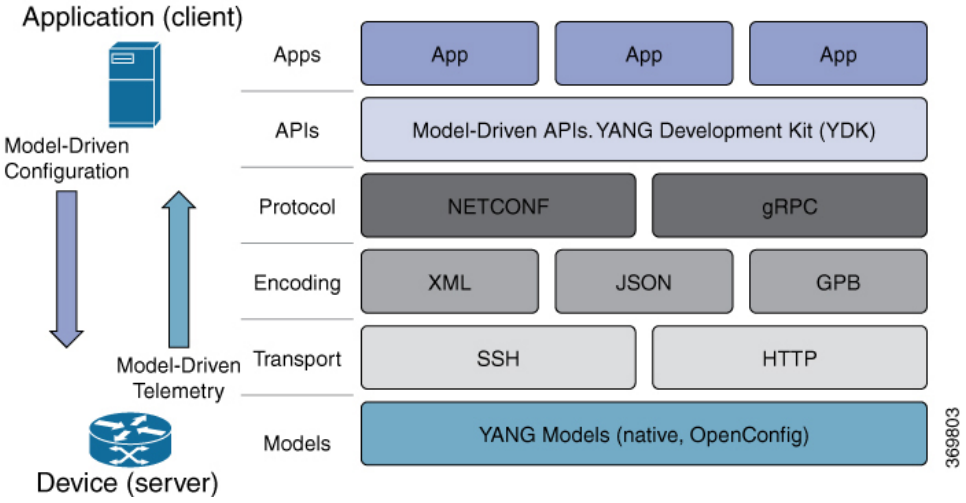
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the general mechanism of CLI scraping is not flexible and optimal. Small changes in the configuration require rewriting scripts multiple times. Bulk configuration changes through CLIs are cumbersome and error-prone. These limitations restrict automation and scale. To overcome these limitations, you need an automated mechanism to manage your network.

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

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

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

Figure 1: Model-driven Programmability Layers



Data models provides access to the capabilities of the devices in a network using Network Configuration Protocol (NETCONF Protocol) or google-defined Remote Procedure Calls (gRPC Protocol). The operations on the router are carried out by the protocols using YANG models to automate and programme operations in a network.

Benefits of Data Models

Configuring routers using data models overcomes drawbacks posed by traditional router management because the data models:

- Provide a common model for configuration and operational state data, and perform NETCONF actions.
- Use protocols to communicate with the routers to get, manipulate and delete configurations in a network.
- Automate configuration and operation of multiple routers across the network.

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

- YANG Data Model, on page 2
- Access the Data Models, on page 9
- CLI to Yang Mapping Tool, on page 10
- Communication Protocols, on page 11
- YANG Actions, on page 12

YANG Data Model

A YANG module defines a data model through the data of the router, and the hierarchical organization and constraints on that data. Each module is uniquely identified by a namespace URL. The YANG models describe the configuration and operational data, perform actions, remote procedure calls, and notifications for network devices.

The YANG models must be obtained from the router. The models define a valid structure for the data that is exchanged between the router and the client. The models are used by NETCONF and gRPC-enabled applications.

Note gRPC is supported only in 64-bit platforms.

- **Cisco-specific models:** For a list of supported models and their representation, see Native models.
- **Common models:** These models are industry-wide standard YANG models from standard bodies, such as IETF and IEEE. These models are also called Open Config (OC) models. Like synthesized models, the OC models have separate YANG models defined for configuration data and operational data, and actions.

YANG models can be: For a list of supported OC models and their representation, see OC models.

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

For more details about YANG, refer RFC 6020 and 6087.

Data models handle the following types of requirements on routers (RFC 6244):

- **Configuration data:** A set of writable data that is required to transform a system from an initial default state into its current state. For example, configuring entries of the IP routing tables, configuring the interface MTU to use a specific value, configuring an ethernet interface to run at a given speed, and so on.
- **Operational state data:** A set of data that is obtained by the system at runtime and influences the behavior of the system in a manner similar to configuration data. However, in contrast to configuration data, operational state data is transient. The data is modified by interactions with internal components or other systems using specialized protocols. For example, entries obtained from routing protocols such as OSPF, attributes of the network interfaces, and so on.
- **Actions:** A set of NETCONF actions that support robust network-wide configuration transactions. When a change is attempted that affects multiple devices, the NETCONF actions simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.

For more information about Data Models, see RFC 6244.

YANG data models can be represented in a hierarchical, tree-based structure with nodes. This representation makes the models easy to understand.

Each feature has a defined YANG model, which is synthesized from schemas. A model in a tree format includes:

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

YANG defines four node types. Each node has a name. Depending on the node type, the node either defines a value or contains a set of child nodes. The nodes types for data modeling are:

- leaf node - contains a single value of a specific type
- leaf-list node - contains a sequence of leaf nodes

- list node - contains a sequence of leaf-list entries, each of which is uniquely identified by one or more key leaves
- container node - contains a grouping of related nodes that have only child nodes, which can be any of the four node types

Structure of LLDP Data Model

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

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

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

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


  prefix "ethernet-lldp-cfg";

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

  import cisco-semver { prefix "semver"; }

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

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";

  contact
    "Cisco Systems, Inc.
     Customer Service

     Postal: 170 West Tasman Drive
     San Jose, CA 95134

     Tel: +1 800 553-NETS

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

The structure of a data model can be explored using a YANG validator tool such as pyang and the data model can be formatted in a tree structure.

LLDP Configuration Data Model

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

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

```
         +--rw active                          Interface-active
         +--rw interface-name                  xr:Interface-name
         +--rw ethernet-lldp-cfg:lldp!
            +--rw ethernet-lldp-cfg:transmit
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:receive
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:lldp-intf-enter    boolean
            +--rw ethernet-lldp-cfg:enable?            Boolean

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

LLDP Operational Data Model

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

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

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

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

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

Components of a YANG Module

A YANG module defines a single data model. However, a module can reference definitions in other modules and sub-modules by using one of these statements:

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

- **import** imports external modules
- **include** includes one or more sub-modules
- **augment** provides augmentations to another module, and defines the placement of new nodes in the data model hierarchy
- **when** defines conditions under which new nodes are valid
- **prefix** references definitions in an imported module

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

YANG Module Set

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

Accessing the deprecated Cisco IOS XR YANG models generates a syslog message only once until the YANG agent (NETCONF or Emsd) restarts. After a restart, the message is generated again. For deprecated Cisco IOS XR data models, see Backward InCompatible (BIC) folder from Deprecated XPaths.

Configure YANG Module Set

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

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

Access the Data Models

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

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

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

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

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

```
<rpc-reply message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper-sub1</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>Cisco-IOS-XR-snmp-agent-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
    <location>NETCONF</location>
</schema>

------------<snipped>--------------
<schema>
    <identifier>openconfig-aft-types</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/fib-types</namespace>
    <location>NETCONF</location>
</schema>
<schema>
```

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

CLI to Yang Mapping Tool

Table 1: Feature History Table

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

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

CLI commands are widely used for configuring and extracting the operational details of a router. But bulk configuration changes through CLIs are cumbersome and error-prone. These limitations restrict automation and scale. To overcome these limitations, you need an automated mechanism to manage your network. Cisco IOS XR supports a programmatic way of configuring and collecting operational data of a router using Yang data models. However, owing to the large number of CLI commands, it is cumbersome to determine the mapping between the CLI command and its associated data model.

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

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

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

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

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

The tool supports description of both operational and configurational commands.

Example: Configuration Data

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

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

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


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

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

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

Example: Operational Data

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

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

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

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

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

Communication Protocols

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

YANG uses one of these protocols:

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

Note gRPC is supported only in 64-bit platforms.

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

Protocol	Transport	Encoding/ Decoding
NETCONF	ssh	xml
gRPC	http/2	json

NETCONF Protocol

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

gRPC Protocol

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

Note gRPC is supported only in 64-bit platforms.

YANG Actions

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

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

Note NETCONF supports XML format, and gRPC supports JSON format.

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

Actions	YANG Models
logmsg	Cisco-IOS-XR-syslog-act
snmp	Cisco-IOS-XR-snmp-test-trap-act
rollback	Cisco-IOS-XR-cfgmgr-rollback-act
clear isis	Cisco-IOS-XR-isis-act
clear bgp	Cisco-IOS-XR-ipv4-bgp-act

Example: PING NETCONF Action

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

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">
  <destination>
   <destination>1.2.3.4</destination>
  </destination>
 </ping>
</rpc>
```

This section shows the NETCONF action response from the router.

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

Example: XR Process Restart Action

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

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

```
    <location>0/RP0/CPU0</location>
  </sysmgr-process-restart>
</rpc>
```

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

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

Example: Copy Action

This example shows the RPC request and response for `copy` action:

RPC request:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <copy xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">
    <sourcename>//root:<location>/100MB.txt</sourcename>
    <destinationname>/</destinationname>
    <sourcefilesystem>ftp:</sourcefilesystem>
    <destinationfilesystem>harddisk:</destinationfilesystem>
    <destinationlocation>0/RSP1/CPU0</destinationlocation>
  </copy>
</rpc>
```

RPC response:

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">Successfully
completed copy operation</response>
</rpc-reply>

8.261830565s elapsed
```

Example: Delete Action

This example shows the RPC request and response for `delete` action:

RPC request:

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

RPC response:

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">Successfully
 completed delete operation</response>
</rpc-reply>

395.099948ms elapsed
```

Example: Install Action

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

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

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

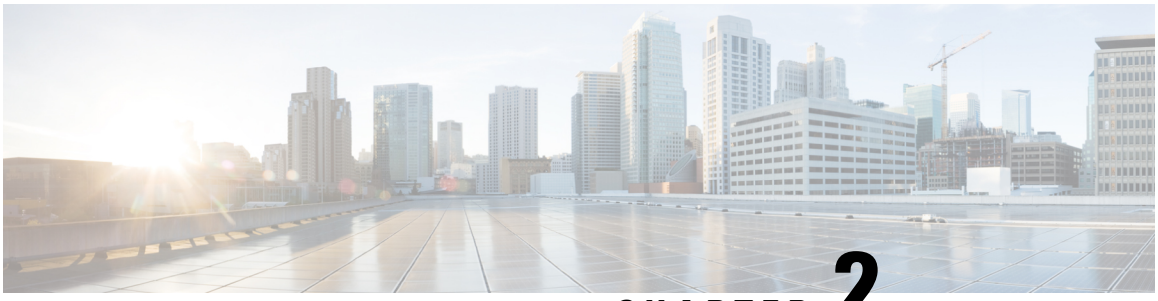
```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <op-id xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-spirit-install-act">6</op-id>
</rpc-reply>
```

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

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

Restrictions for Install Action

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



CHAPTER 2

Use NETCONF Protocol to Define Network Operations with Data Models

Table 2: Feature History Table

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

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

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

NETCONF uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. It uses a simple RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. The client can be a script or application that runs as part of a network manager. The server is a network device such as a router. NETCONF defines how to communicate with the devices, but does not handle what data is exchanged between the client and the server.

Note Accessing the router via NETCONF grants by default write permissions for a user, in spite of read-only access configured for this user for CLI access, as CLI authorization is bypassed.

NETCONF Session

A NETCONF session is the logical connection between a network configuration application (client) and a network device (router). The configuration attributes can be changed during any authorized session; the effects are visible in all sessions. NETCONF is connection-oriented, with SSH as the underlying transport. NETCONF sessions are established with a `hello` message, where features and capabilities are announced. At the end of each message, the NETCONF agent sends the `]]>]]>` marker. Sessions are terminated using `close` or `kill` messages.

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

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

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

```
netconf-xml agent listens on port 22

netconf-yang agent listens on port 830
```

Version 1.0 The NETCONF XML agent accepts the message.

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

Version 1.1 The NETCONF YANG agent accepts the message.

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

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

Example:

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

```
          <neighbors/>
          <instance-name/>
        </instance>
      </instances>
    </isis>
  </filter>
</get>
</rpc>

##
```

Configure NETCONF Agent

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

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

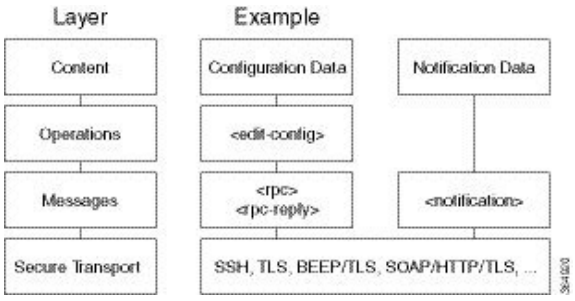
To enable the NETCONF SSH agent, use the following command:

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

NETCONF Layers

NETCONF protocol can be partitioned into four layers:

Figure 2: NETCONF Layers



- **Content layer:** includes configuration and notification data
- **Operations layer:** defines a set of base protocol operations invoked as RPC methods with XML-encoded parameters
- **Messages layer:** provides a simple, transport-independent framing mechanism for encoding RPCs and notifications
- **Secure Transport layer:** provides a communication path between the client and the server

For more information about NETCONF, refer RFC 6241.

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

- NETCONF Operations, on page 20
- Retrieve Default Parameters Using with-defaults Capability, on page 24
- Retrieve Transaction ID for NSO Operations, on page 30

• Set Router Clock Using Data Model in a NETCONF Session, on page 32

NETCONF Operations

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

The base protocol includes the following NETCONF operations:

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

These NETCONF operations are described in the following table:

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

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

NETCONF Operation	Description	Example

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

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

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

NETCONF Operation	Description	Example

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

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

NETCONF Operation to Get Configuration

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

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

Netconf Request (Client to Router)	Netconf Response (Router to Client)

`<rpc message-id="101"`
`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
`<get-config>`
`<source><running/></source>`
`<filter>`
`<lldp`
`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-cfg"/>`
 	
`</filter>`
`</get-config>`
`</rpc>`
	`<?xml version="1.0"?>`
	`<rpc-reply message-id="101"`
	`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
	`<data>`
	`<lldp`
	`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-cfg">`
	
	`<timer>60</timer>`
	`<enable>true</enable>`
	`<reinit>3</reinit>`
	`<holdtime>150</holdtime>`
	`</lldp>`
	
	`</data>`
	`</rpc-reply>`
	`319 bytes received`
	`6.409561ms elapsed`

The <rpc> element in the request and response messages enclose a NETCONF request sent between the client and the router. The `message-id` attribute in the <rpc> element is mandatory. This attribute is a string chosen by the sender and encodes an integer. The receiver of the <rpc> element does not decode or interpret this string but simply saves it to be used in the <rpc-reply> message. The sender

must ensure that the `message-id` value is normalized. When the client receives information from the server, the <rpc-reply> message contains the same `message-id`.

Retrieve Default Parameters Using with-defaults Capability

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

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

Cisco IOS XR routers support only the explicit basic mode. A server that uses this mode must consider any data node that isn't explicitly set to be the default data.

As per RFC 6243, the router supports `<with-defaults>` capability to retrieve the default parameters of configuration and state data node using a NETCONF protocol operation. The `<with-defaults>` capability indicates which default-handling basic mode is supported by the server. It also indicates support for additional retrieval modes. These retrieval modes allow a NETCONF client to control whether the server returns the default data.

By default, `<with-defaults>` capability is disabled. To enable this capability, use the following command in Config mode:

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

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

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

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

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

Example 1: Create Operation

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

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

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

```
<enabled xc:operation="create">false</enabled>
<index xc:operation="create">2</index>
</config>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</config>
</edit-config>
</rpc>
```

Response received from the NETCONF agent:

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

Commit the configuration.

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

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

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

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

```
[host 172.x.x.x session-id 2985924161] Received:
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:1f29267f-7593-4a3c-8382-6ab9bec323ca"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
   <error-tag>data-exists</error-tag>
   <error-severity>error</error-severity>
   <error-path
xmlns:ns1="http://openconfig.net/yang/interfaces">ns1:interfaces/ns1:interface[name =
'TenGigE0/0/0/0']/ns1:subinterfaces/ns1:subinterface[index = '2']/ns1:config</error-path>
 </rpc-error>
</rpc-reply>
```

Example 2: Delete Operation

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

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

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

Response received from the NETCONF agent:

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

Example 3: Copy Configuration

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

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

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
 <target>
  <candidate/>
 </target>
 <source>
 <config>
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
    <name>TenGigE0/0/0/0</name>
    <subinterfaces>
     <subinterface>
      <index>2</index>
      <config>
       <index>2</index>
      </config>
     </subinterface>
    </subinterfaces>
   </interface>

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

The show run command shows the copied configuration.

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

Example 4: Get Configuration

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

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

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

```
 </get-config>
 </rpc>
```

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

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:99d8b2d0-ab05-474a-bc02-9242ba511308"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
     <name>TenGigE0/0/0/0</name>
       <subinterfaces>
       <subinterface>
         <index>2</index>
           <config>
            <index>2</index>
            <enabled>false</enabled>
           </config>
         <ipv6 xmlns="http://openconfig.net/yang/interfaces/ip">
         <config>
            <enabled>false</enabled>
           </config>
          </ipv6>
        </subinterface>
    </subinterfaces>
   </interface>
 <interface>
  <name>MgmtEth0/RSP0/CPU0/0</name>
   <config>
    <name>MgmtEth0/RSP0/CPU0/0</name>
     <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:ethernetCsmacd</type>

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

```
    <config>
      <auto-negotiate>false</auto-negotiate>
    </config>
  </ethernet>
 </interface>
 </interfaces>
 </data>
</rpc-reply>
READ operation completed
```

Example 5: Get Operation

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

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

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

<get> response received from the NETCONF agent:

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

```
      <out-unicast-pkts>0</out-unicast-pkts>
      <out-broadcast-pkts>0</out-broadcast-pkts>
      <out-multicast-pkts>0</out-multicast-pkts>
      <out-pkts>0</out-pkts>
      <out-octets>0</out-octets>
      <out-discards>0</out-discards>
      <in-discards>0</in-discards>
      <in-unknown-protos>0</in-unknown-protos>
      <in-errors>0</in-errors>
      <in-fcs-errors>0</in-fcs-errors>
      <out-errors>0</out-errors>
     <carrier-transitions>0</carrier-transitions>
    <last-clear>2020-03-02T15:35:30.927+00:00</last-clear>
   </counters>
  <ifindex>92</ifindex>
   <logical>true</logical>
  </state>
 </subinterface>
 </subinterfaces>
 </interface>
 </interfaces>
 </data>
</rpc-reply>
READ operation completed
```

Retrieve Transaction ID for NSO Operations

Table 3: Feature History Table

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

Cisco Network Services Orchestrator (NSO) is a data model-driven platform for automating your network orchestration. NSO uses NETCONF-based Network Element Drivers (NED) to synchronize the configuration

states of the routers it manages. NEDs comprise of the network-facing part of NSO and communicate over the native protocol supported by the router, such as Network Configuration Protocol (NETCONF).

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

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

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

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

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

Where:

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

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

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

```
      </global>
    </config-manager>
 </filter>
</get>
</rpc>


Response:
<rpc-reply message-id="test" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <config-manager xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-oper">
   <global>
    <config-commit>
     <last-commit-id
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper">
        XR:1000000009;Admin:1595-891537-949905</last-commit-id>
    </config-commit>
   </global>
  </config-manager>
 </data>
</rpc-reply>
```

Set Router Clock Using Data Model in a NETCONF Session

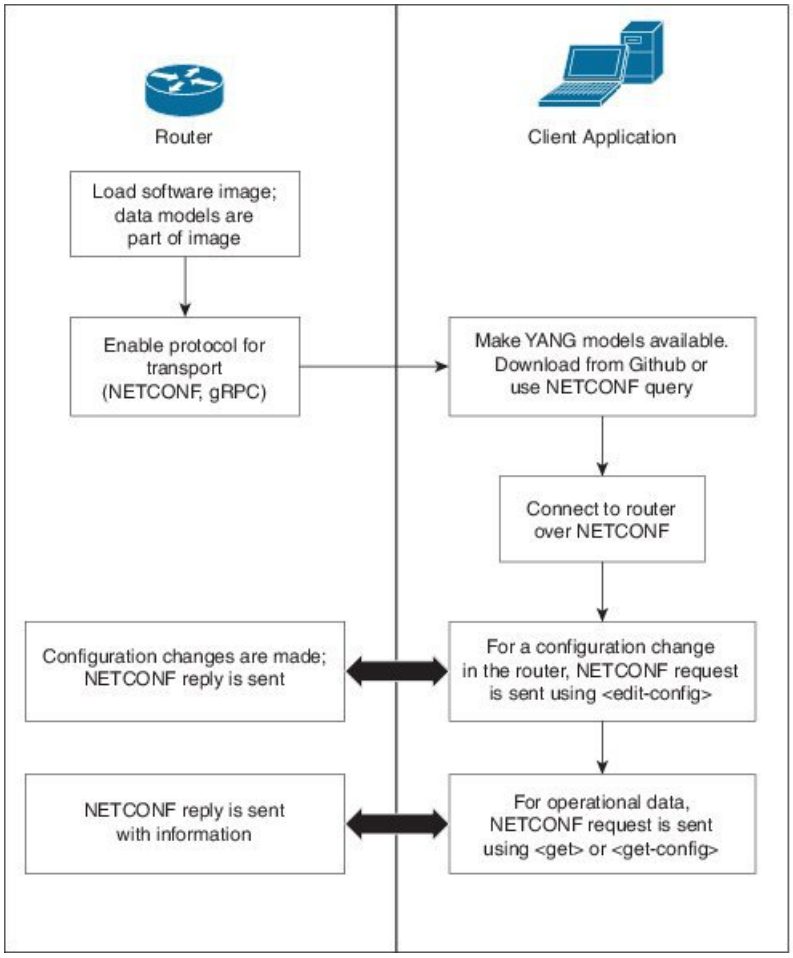
The process for using data models involves:

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

Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization using **aaa authorization exec** command before setting up any configuration. For more information about configuring AAA authorization, see the *System Security Configuration Guide*.

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

Figure 3: Process for Using Data Models

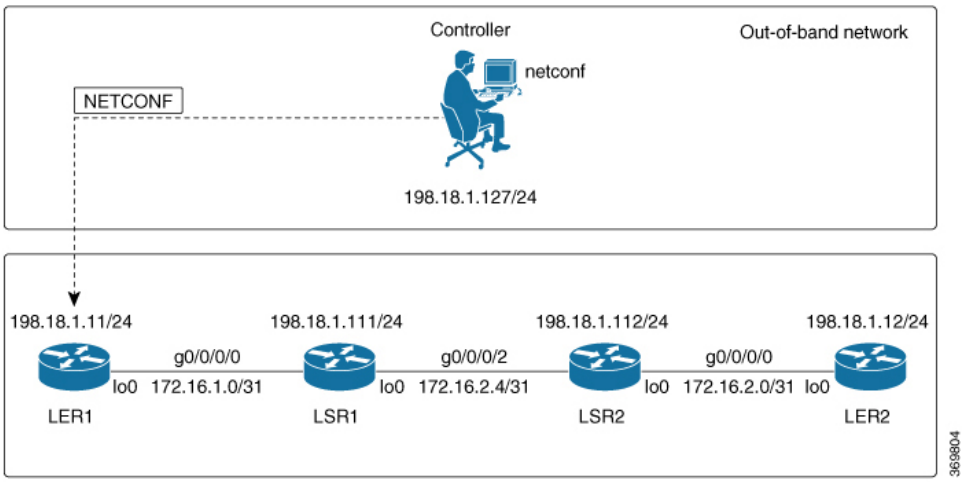


In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

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

The following image illustrates the network topology:

Figure 4: Network Topology for gRPC session



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

Before you begin

Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information

Procedure

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

Example:

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

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

Example:

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

```
        | +--ro time-zone? string
        | +--ro time-source? Time-source
        +--ro uptime
            +--ro host-name? string
            +--ro uptime? uint32
```

Step 3 Retrieve the current time on router LER1.

Example:

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

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

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

Example:

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
  <system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

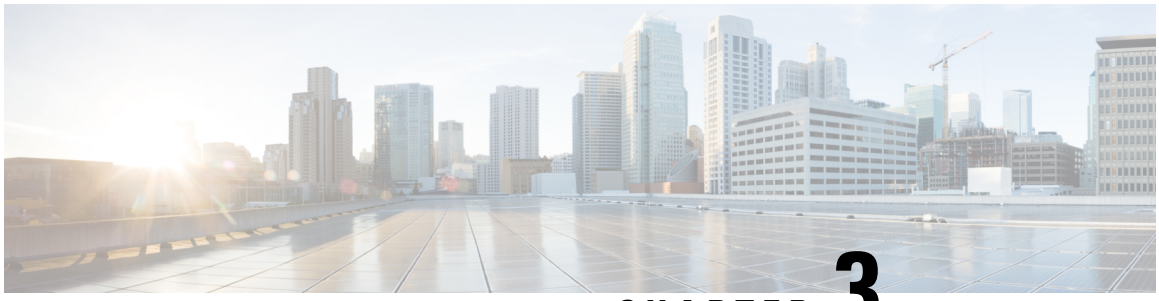
Step 5 Verify that the router clock is set to PST time zone.

Example:

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

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

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



CHAPTER 3

Use gRPC Protocol to Define Network Operations with Data Models

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

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

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

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

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

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

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

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

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

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

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

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

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

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

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

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

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

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

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

}

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

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

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

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

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

Example for gRPCExec configuration:

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

}

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

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

Example for OpenConfiggRPC configuration:

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

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

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

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

- gRPC Operations, on page 40
- gRPC Network Management Interface, on page 41
- gRPC Network Operations Interface , on page 41
- Configure Interfaces Using Data Models in a gRPC Session, on page 41

gRPC Operations

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

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

gRPC Operation to Get Configuration

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

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

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

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

gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

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

gRPC Network Operations Interface

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

Configure Interfaces Using Data Models in a gRPC Session

Google-defined remote procedure call () is an open-source RPC framework. gRPC supports IPv4 and IPv6 address families. The client applications use this protocol to request information from the router, and make configuration changes to the router.

The process for using data models involves:

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

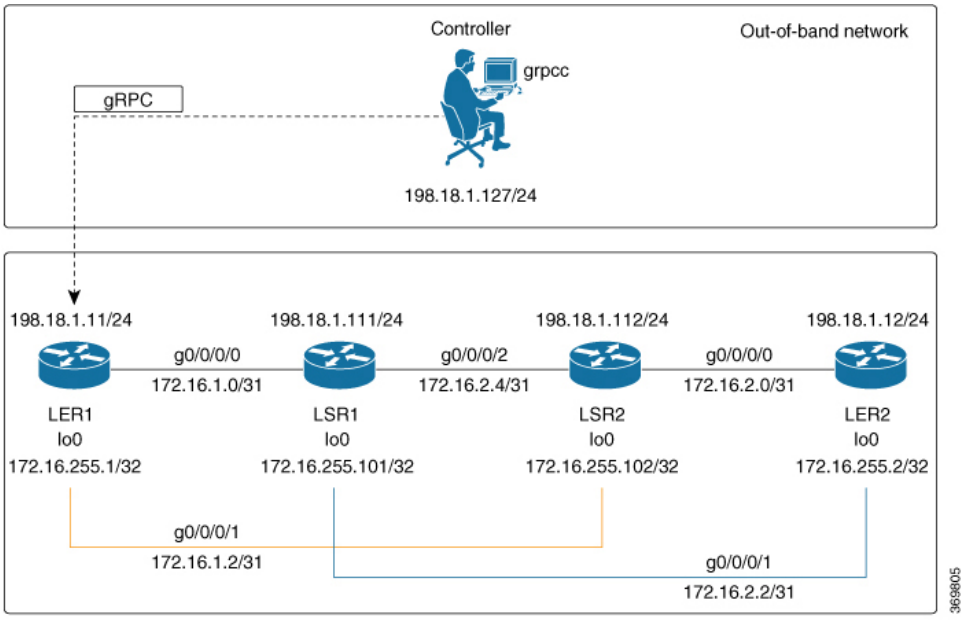
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization using **aaa authorization exec** command before setting up any configuration. For more information about configuring AAA authorization, see the *System Security Configuration Guide*.

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

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

The following image illustrates the network topology:

Figure 5: Network Topology for gRPC session



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

Before you begin

- Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information
- Configure Transport Layer Security (TLS). Enabling gRPC protocol uses the default HTTP/2 transport with no TLS. gRPC mandates AAA authentication and authorization for all gRPC requests. If TLS is not configured, the authentication credentials are transferred over the network unencrypted. Enabling TLS ensures that the credentials are secure and encrypted. Non-TLS mode can only be used in secure internal network.

Procedure

Step 1 Enable gRPC Protocol

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

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

a) Enable gRPC over an HTTP/2 connection.

Example:

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

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

b) Set the session parameters.

Example:

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

where:

- `address-family`: set the address family identifier type.
- `certificate-authentication`: enables certificate based authentication
- `dscp`: set QoS marking DSCP on transmitted gRPC.
- `max-request-per-user`: set the maximum concurrent requests per user.
- `max-request-total`: set the maximum concurrent requests in total.
- `max-streams`: set the maximum number of concurrent gRPC requests. The maximum subscription limit is 128 requests. The default is 32 requests.
- `max-streams-per-user`: set the maximum concurrent gRPC requests for each user. The maximum subscription limit is 128 requests. The default is 32 requests.
- `no-tls`: disable transport layer security (TLS). The TLS is enabled by default

- `tlsv1-disable`: disable TLS version 1.0
- `service-layer`: enable the grpc service layer configuration.

  This parameter is not supported in Cisco ASR 9000 Series Routers, Cisco NCS560 Series Routers, , and Cisco NCS540 Series Routers.
- `tls-cipher`: enable the gRPC TLS cipher suites.
- `tls-mutual`: set the mutual authentication.
- `tls-trustpoint`: configure trustpoint.
- `server-vrf`: enable server vrf.

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

Step 2 Configure the interfaces.

In this example, you configure interfaces using Cisco IOS XR native model `Cisco-IOS-XR-ifmgr-cfg.yang`. You gain an understanding about the various gRPC operations while you configure the interface. For the complete list of operations, see gRPC Operations, on page 40. In this example, you merge configurations with `merge-config` RPC, retreive operational statistics using `get-oper` RPC, and delete a configuration using `delete-config` RPC. You can explore the structure of the data model using YANG validator tools such as pyang.

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

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

a) Explore the XR configuration model for interfaces and its IPv4 augmentation.

Example:

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

b) Configure a loopback0 interface on LER1.

Example:

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

c) Merge the configuration.

Example:

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

d) Configure the ethernet interface on LER1.

Example:

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

e) Merge the configuration.

Example:

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

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

Example:

```
controller:grpc$ grpcc -username admin -password admin -oper delete-config
-server_addr 198.18.1.11:57400 -yang_path "$(< xr-interfaces-gi0-shutdown-cfg.json )"
emsDeleteConfig: Sending ReqId 1, yangJson {
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
   "interface-configuration": [
   {
    "active": "act",
    "interface-name": "GigabitEthernet0/0/0/0",
    "shutdown": [
      null
    ]
   }
  ]
 }
}
emsDeleteConfig: Received ReqId 1, Response ''
```

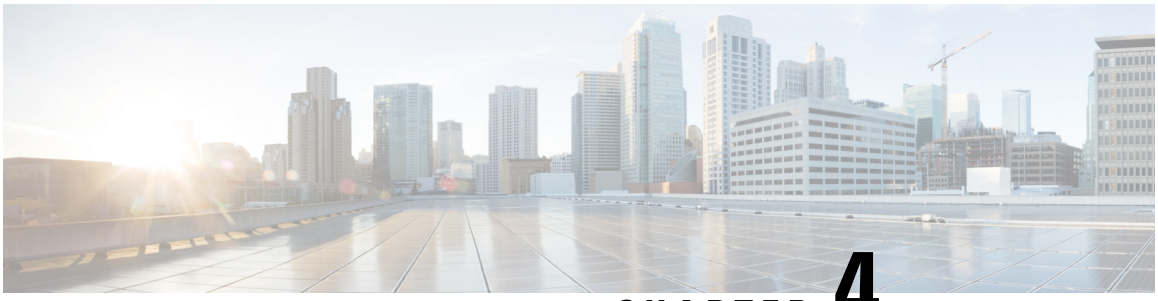
Step 3 Verify that the loopback interface and the ethernet interface on router LER1 are operational.

Example:

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

```
        "interface-name": "GigabitEthernet0/0/0/1",
        "interface": "GigabitEthernet0/0/0/1",
        "type": "IFT_GETHERNET",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "ether",
        "encapsulation-type-string": "ARPA",
        "mtu": 1514,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 1000000
       },
       {
        "interface-name": "Loopback0",
        "interface": "Loopback0",
        "type": "IFT_LOOPBACK",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "loopback",
        "encapsulation-type-string": "Loopback",
        "mtu": 1500,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 0
    },
    {
        "interface-name": "MgmtEth0/RP0/CPU0/0",
        "interface": "MgmtEth0/RP0/CPU0/0",
        "type": "IFT_ETHERNET",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "ether",
        "encapsulation-type-string": "ARPA",
        "mtu": 1514,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 1000000
    },
    {
        "interface-name": "Null0",
        "interface": "Null0",
        "type": "IFT_NULL",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "null",
        "encapsulation-type-string": "Null",
        "mtu": 1500,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 0
      }
    ]
   }
  }
 }
}
emsGetOper: ReqId 1, byteRecv: 2325
```

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



CHAPTER 4

Unified Configuration Models

Table 4: Feature History Table

Feature Name	Release Information	Description
Unified Data Model to map script file to the custom OID	Release 7.5.3	Use the `Cisco-IOS-XR-um-script-server-cfg.yang` unified data model to map script file to the custom OID.
New Unified Models	Release 7.5.2	Unified models are CLI-based YANG models that are designed to replace the native schema-based models. This release introduces new unified models to configure the Fabric Interface ASIC (FIA), Link Aggregation Control Protocol (LACP), Cisco Express Forwarding (CEF) and controller fabric. You can access these new unified models from the Github repository.
Transitioning Native Models to Unified Models (UM)	Release 7.4.1	Unified models are CLI-based YANG models that are designed to replace the native schema-based models. UM models are generated directly from the IOS XR CLIs and mirror them in several ways. This results in improved usability and faster adoption of YANG models. You can access the new unified models from the Github repository.

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

Table 5: Unified Models

Unified Models	Introduced in Release
Cisco-IOS-XR-um-script-server-cfg	Release 7.5.3
Cisco-IOS-XR-um-script-cfg	Release 7.5.3

Unified Models	Introduced in Release
Cisco-IOS-XR-um-cont-cpri-cfg	Release 7.5.2
Cisco-IOS-XR-um-hw-module-service-cfg	Release 7.5.2
Cisco-IOS-XR-um-lacp-cfg	Release 7.5.2
Cisco-IOS-XR-um-cef-pd-cfg	Release 7.5.2
Cisco-IOS-XR-um-if-ipsubscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-session-redundancy-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-accounting-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-subscriber-redundancy-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-opendns-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-cfg	Release 7.5.1
Cisco-IOS-XR-um-lpts-profiling-cfg	Release 7.5.1
Cisco-IOS-XR-um-ppp-cfg	Release 7.5.1
Cisco-IOS-XR-um-pppoe-cfg	Release 7.5.1
Cisco-IOS-XR-um-vpdn-cfg	Release 7.5.1
Cisco-IOS-XR-um-aaa-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-ipv4-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-ipv6-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-vrf-cfg	Release 7.5.1
Cisco-IOS-XR-um-mibs-subscriber-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-monitor-session-cfg	Release 7.5.1
Cisco-IOS-XR-um-l2tp-class-cfg	Release 7.5.1
Cisco-IOS-XR-um-dynamic-template-dhcpv6d-cfg	Release 7.5.1
Cisco-IOS-XR-um-dyn-tmpl-service-policy-cfg	Release 7.5.1
Cisco-IOS-XR-um-snmp-server mroutemib send-all-cfg	Release 7.5.1
Cisco-IOS-XR-um-aaa-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-diameter-cfg	Release 7.4.1

Unified Models	Introduced in Release
Cisco-IOS-XR-um-aaa-nacm-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-tacacs-server-cfg	Release 7.4.1
Cisco-IOS-XR-um-aaa-task-user-cfg	Release 7.4.1
Cisco-IOS-XR-um-banner-cfg	Release 7.4.1
Cisco-IOS-XR-um-bfd-sbfd-cfg	Release 7.4.1
Cisco-IOS-XR-um-call-home-cfg	Release 7.4.1
Cisco-IOS-XR-um-cdp-cfg	Release 7.4.1
Cisco-IOS-XR-um-cef-accounting-cfg	Release 7.4.1
Cisco-IOS-XR-um-cfg-mibs-cfg	Release 7.4.1
Cisco-IOS-XR-um-cli-alias-cfg	Release 7.4.1
Cisco-IOS-XR-um-clock-cfg	Release 7.4.1
Cisco-IOS-XR-um-config-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-breakout-cfg	Release 7.4.1
Cisco-IOS-XR-um-cont-optics-cfg	Release 7.4.1
Cisco-IOS-XR-um-control-plane-cfg	Release 7.4.1
Cisco-IOS-XR-um-crypto-cfg	Release 7.4.1
Cisco-IOS-XR-um-domain-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-cfm-cfg	Release 7.4.1
Cisco-IOS-XR-um-ethernet-oam-cfg	Release 7.4.1
Cisco-IOS-XR-um-exception-cfg	Release 7.4.1
Cisco-IOS-XR-um-flowspec-cfg	Release 7.4.1
Cisco-IOS-XR-um-frequency-synchronization-cfg	Release 7.4.1
Cisco-IOS-XR-um-hostname-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-port-range-cfg	Release 7.4.1
Cisco-IOS-XR-um-hw-module-profile-cfg	Release 7.4.1
Cisco-IOS-XR-um-ip-virtual-cfg	Release 7.4.1
Cisco-IOS-XR-um-ipsla-cfg	Release 7.4.1
Cisco-IOS-XR-um-l2vpn-cfg	Release 7.4.1

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

Unified Models	Introduced in Release
Cisco-IOS-XR-um-telnet-cfg	Release 7.4.1
Cisco-IOS-XR-um-tpa-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-bridgemib-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-config-copy-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-redundancy-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-entity-state-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-flash-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-fru-ctrl-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-ipsec-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-l2tun-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-otn-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-power-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-selective-vrf-download-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-syslog-cfg	Release 7.4.1
Cisco-IOS-XR-um-traps-system-cfg	Release 7.4.1
Cisco-IOS-XR-um-udp-cfg	Release 7.4.1
Cisco-IOS-XR-um-vty-pool-cfg	Release 7.4.1
Cisco-IOS-XR-um-xml-agent-cfg	Release 7.4.1
Cisco-IOS-XR-um-conflict-policy-cfg	Release 7.3.1
Cisco-IOS-XR-um-flow-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-access-group-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv4-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-ipv6-cfg	Release 7.2.1
Cisco-IOS-XR-um-if-service-policy-qos-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv4-access-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-ipv6-access-list-cfg	Release 7.2.1
Cisco-IOS-XR-um-l2-ethernet-cfg	Release 7.2.1

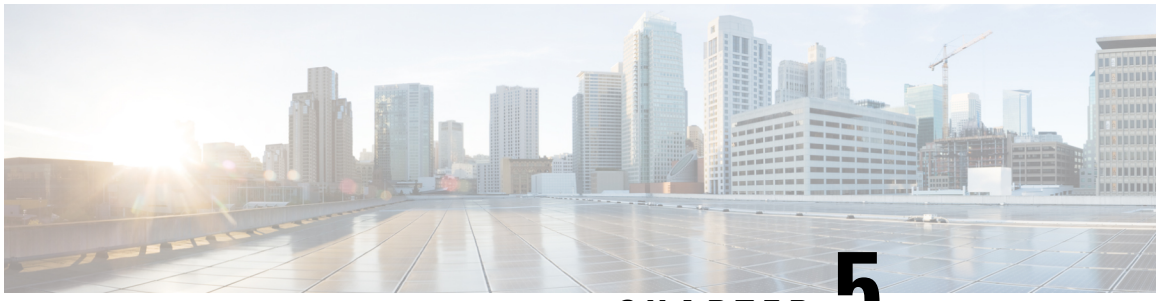
Unified Models	Introduced in Release
Cisco-IOS-XR-um-multicast-routing-cfg	Release 7.2.1
Cisco-IOS-XR-um-object-group-cfg	Release 7.2.1
Cisco-IOS-XR-um-policymap-classmap-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-igmp-cfg	Release 7.2.1
Cisco-IOS-XR-um-router-pim-cfg	Release 7.2.1
Cisco-IOS-XR-um-statistics-cfg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

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

PART I

Automation Scripts

- Achieve Network Operational Simplicity Using Automation Scripts, on page 59
- Config Scripts, on page 61
- Exec Scripts, on page 77
- Process Scripts, on page 93



CHAPTER 5

Achieve Network Operational Simplicity Using Automation Scripts

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

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

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

Note You can create scripts using Python 3.5.

• Explore the Types of Automation Scripts, on page 59

Explore the Types of Automation Scripts

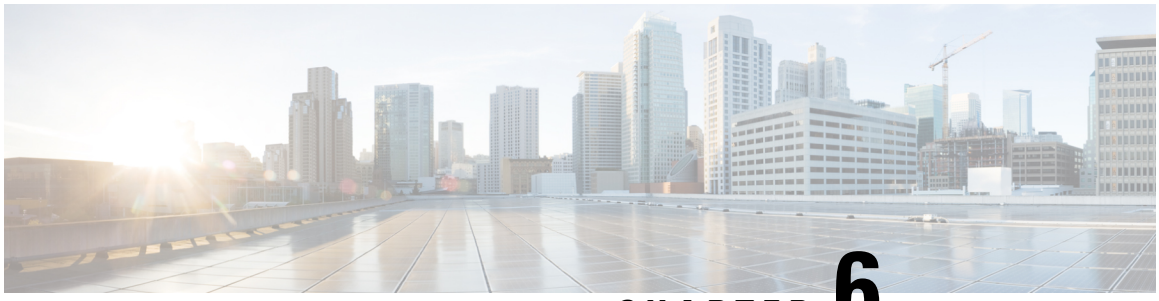
There are four types of on-box automation scripts that you can leverage to automate your network operations:

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

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

Table 6: On-Box Automation Scripts

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



CHAPTER 6

Config Scripts

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

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

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

Config Scripts Limitations

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

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

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

Get Started with Config Scripts

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

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

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

- Workflow to Run Config Scripts, on page 62
- Manage Scripts, on page 69
- Example: Validate and Activate an SSH Config Script, on page 71

Workflow to Run Config Scripts

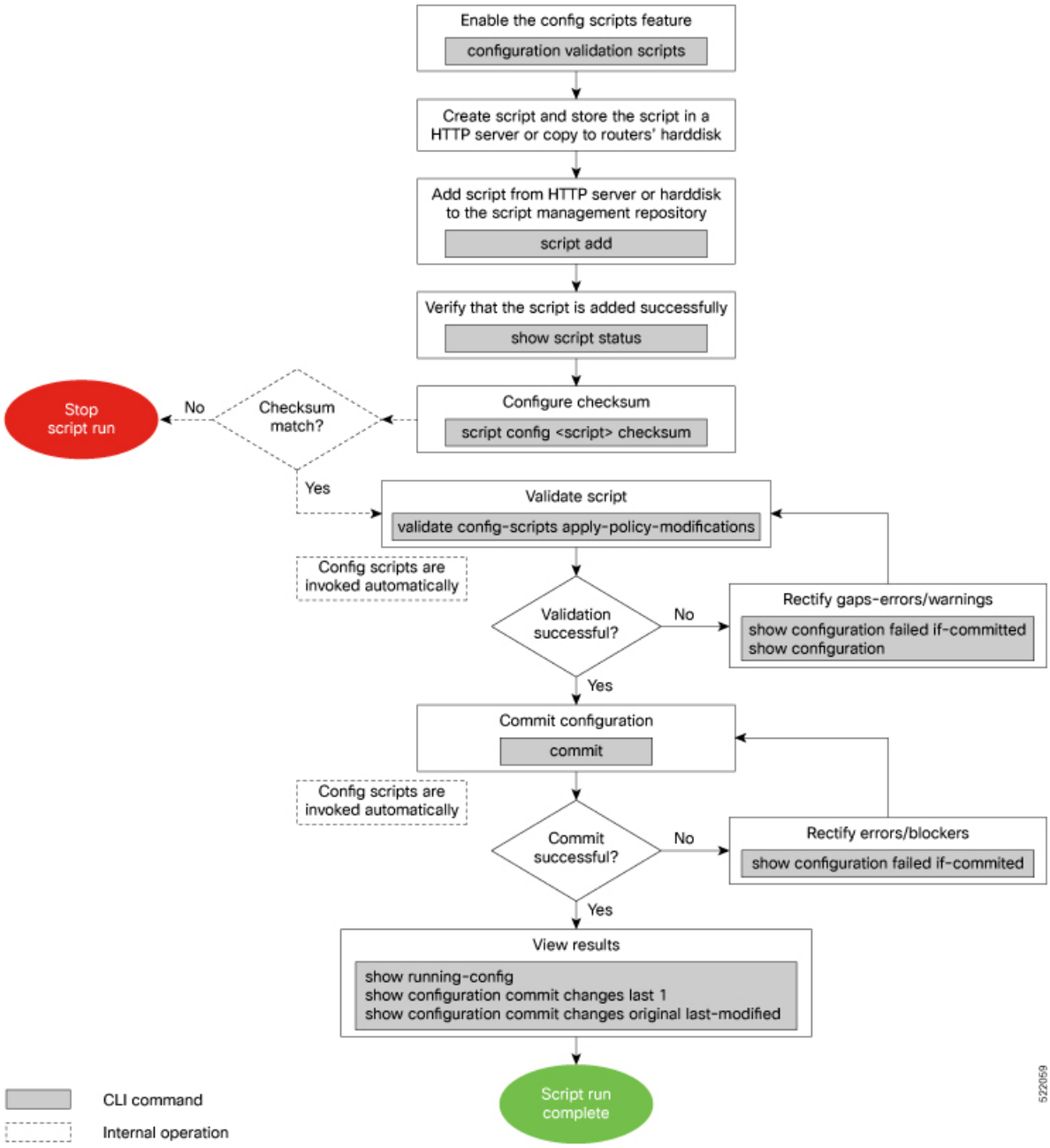
Complete the following tasks to provision config scripts:

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

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

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

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



Enable Config Scripts Feature

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

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

Procedure

Step 1 Enable the config scripts.

Example:

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

Step 2 Commit the configuration.

Example:

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

Download the Script to the Router

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

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

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

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

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

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 a configured HTTP server or the harddisk location in the router.

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

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

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

You can add a maximum of 10 scripts simultaneously.

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

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

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

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

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

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

Note
Only SHA256 checksum is supported.

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

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

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

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

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

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

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

Example:

```
Router#show script status
Router#show script status
Wed Aug 25 23:10:50.453 UTC
=============================================================================================
 Name              | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
 config-script.py  | config   | Config Checksum  | NEW         | Tue Aug 24 10:18:23 2021
```

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

Configure Checksum for Config Script

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

script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a syslog warning message is displayed.

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

Note Config scripts support SHA256 checksum.

Before you begin

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

1. Enable Config Scripts Feature, on page 63
2.

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
Fri Aug 20 05:04:13.539 UTC
=====================================================================================================

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

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

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

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

Step 3 Configure the checksum.

Example:

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

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

If you are configuring multiple scripts, the system decides an appropriate order to run the scripts. However, you can control the order in which scripts execute using a priority value. For more information on configuring the priority value, see Control Priority When Running Multiple Scripts, on page 70.

Step 4 Verify the status of the script.

Example:

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

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

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

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

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

Validate or Commit Configuration to Invoke Config Script

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

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

Before you begin

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

1. Enable Config Scripts Feature, on page 63
2. Configure Checksum for Config Script, on page 65

Procedure

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

Example:

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

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

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

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

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

Step 2 View the modified target configuration.

Example:

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

Step 3 Commit the configuration.

Example:

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

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

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

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

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

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

Example:

```
Router(config)#show logging
Tue Aug 31 08:31:54.472 UTC
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: Disabled
    Monitor logging: level debugging, 0 messages logged
    Trap logging: level informational, 0 messages logged
    Buffer logging: level debugging, 13 messages logged

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

Manage Scripts

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

Delete Config Script from the Router

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

Procedure

Step 1 View the active scripts on the router.

Example:

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

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

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

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

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

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

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

Step 2 Delete script `ssh_config_script.py`.

Example:

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

You can also delete multiple scripts simultaneously.

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

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

Example:

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

The script is deleted from the script management repository.

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

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

Control Priority When Running Multiple Scripts

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

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

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

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

Procedure

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

Example:

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

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

Example:

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

Step 3 Commit the configuration.

Example:

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

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

Example: Validate and Activate an SSH Config Script

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

Before you begin

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

1. Enable config scripts feature on the router. See Enable Config Scripts Feature, on page 63.
2. Create a config script `ssh_config_script.py`. Store the script on an HTTP server or copy the script to the harddisk of the router.

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

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

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

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

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

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

The script checks the following actions:

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

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

Scenario 1: Validate the Script Without SSH Configuration

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

Procedure

Step 1 Configure the checksum to verify the authenticity and integrity of the script. See Configure Checksum for Config Script, on page 65.

Step 2 Validate the config script.

Example:

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

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

The validation of the configuration failed.

Step 3 View the configuration of the failed operation.

Example:

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

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

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

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

Example:

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

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

Scenario 2: Configure SSH and Validate the Script

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

Procedure

Step 1 Configure SSH.

Example:

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

Step 2 Configure the checksum.

Step 3 Validate the configuration again.

Example:

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

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

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

Step 4 Commit the configuration.

Example:

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

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

Example:

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

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

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

Example:

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

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

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

In this example, you see the response after setting the rate-limit to the default value configured in the script. The rate-limit is used to limit the incoming SSH connection requests to the configured rate. The SSH server rejects any connection request beyond the rate-limit. Changing the rate-limit does not affect established SSH sessions. For example, if the rate-limit argument is set to 60, then 60 requests are allowed per minute. The script checks if the rate-limit is set to the default value 60. If yes, the script displays a warning message that the `RATE_LIMIT should not be set to default value`, but allow the commit operation.

Procedure

Step 1 Configure rate-limit to the default value of 60.

Example:

```
Router(config)#ssh server rate-limit 60
```

Step 2 Commit the configuration.

Example:

```
Router(config)#commit
Wed Sep 1 22:11:05.448 UTC

% Validation warnings detected as a result of the commit operation.
Please issue 'show configuration warnings' to view the warnings
```

The script displays a warning message but proceeds with the commit operation.

Step 3 View the warning message.

Example:

```
Router(config)#show configuration warnings
Wed Sep 1 22:12:05.448 UTC
!! SEMANTIC ERRORS: This configuration was rejected by the system due to
semantic errors. The individual errors with each failed configuration command
can be found below.

script config ssh_config_script.py checksum SHA256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
!!% WARNING: RATE_LIMIT should not be set to default value
end
```

The rate limit is default value of 60. The script is programmed to display a warning message if the rate limit is set to the default value. You can either change the limit or leave the value as is.

Step 4 View the running configuration.

Example:

```
Router(config)#do show running-config  script
Wed Sep 1 22:15:05.448 UTC
script config ssh_config_script.py checksum SHA256
2b061f11ede3c1c0c18f1ee97269fd342adb35cbc8a0cd4b6ea1063d0eda2d58
```

The script `ssh_config_script.py` is active.

Scenario 4: Delete SSH Server Configuration

In this example, you delete the SSH server configurations, and see the response when the script is validated.

Procedure

Step 1 Remove the SSH server configuration.

Example:

```
Router(config)#no ssh server v2
```

Step 2 Commit the configuration.

Example:

```
Router(config)#commit
Wed Sep 1 22:45:05.559 UTC

% Failed to commit one or more configuration items during an atomic operation.
No changes have been made. Please issue 'show configuration failed if-committed' from
this session to view the errors
```

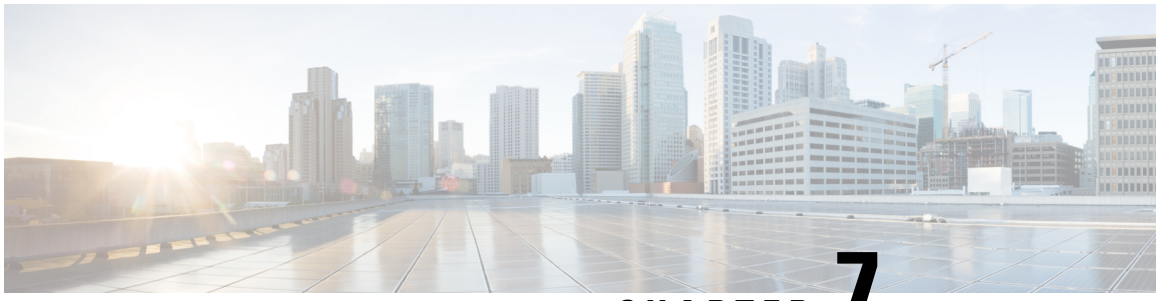
Step 3 View the error message.

Example:

```
Router(config)#show configuration failed if-committed
Wed Sep 1 22:47:53.202 UTC
!! SEMANTIC ERRORS: This configuration was rejected by the system due to semantic errors. The individual
errors with each failed configuration command can be found below.

no ssh server v2
!!% ERROR: Server V2 need to be set
end
```

The message is displayed based on the rule set in the script.



CHAPTER 7

Exec Scripts

Cisco IOS XR exec scripts are on-box scripts that automate configurations of devices in the network. The exec scripts are written in Python using the Python libraries that Cisco provides with the base package. For the list of supported packages

A script management repository on the router manages the exec scripts. This repository is replicated on both RPs.

In IOS XR, AAA authorization controls the user access and privileges to perform operations. To run the exec script, you must have root user permissions.

Exec scripts provide the following advantages:

- Provides automation capabilities to simplify complex operations.
- Create customized operations based on the requirement.
- Provide flexibility in changing the input parameters for every script run. This fosters dynamic automation of operational information.
- Detect and display errors and warnings when executing an operation.
- Run multiple automated operations in parallel without blocking the console.

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

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

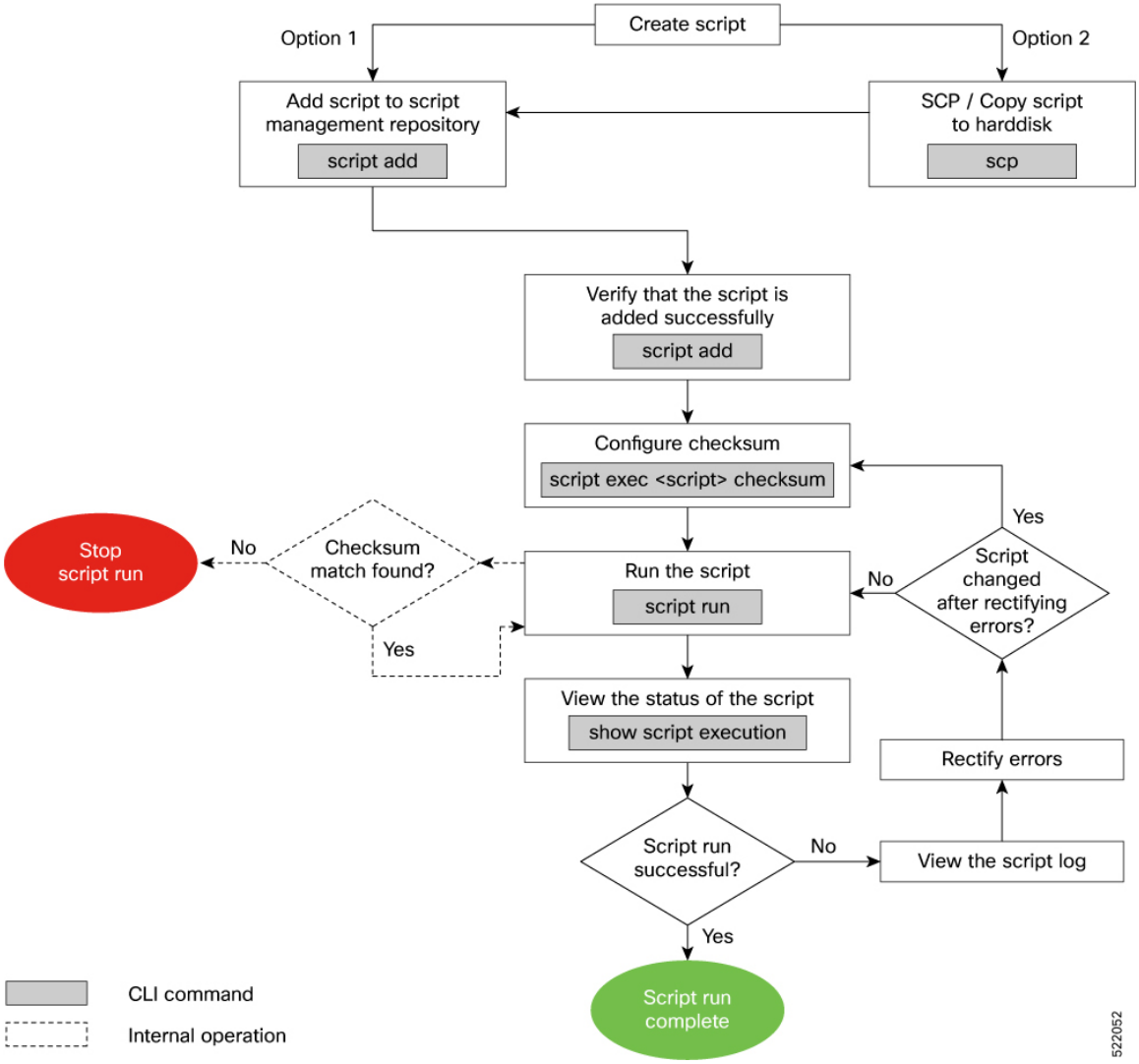
- Workflow to Run an Exec Script, on page 77
- Manage Scripts, on page 85
- Example: Exec Script to Verify Bundle Interfaces, on page 86

Workflow to Run an Exec Script

Complete the following tasks to provision exec scripts:

- Download the script—Add the script to the appropriate exec script directory on the router. using the **script add exec** command.
- Configure checksum—Check script integrity and authenticity using the **script exec <script.py> checksum** command.
- Run the script—Trigger changes to the router configuration. Include arguments, set the maximum time for the script to run, setup log levels using the **script run** command.
- View the script execution details—Validate the script and retrieve the operational data using the **show script execution** command.

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



Download the Script to the Router

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

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

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

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

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

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 a configured HTTP server or the harddisk location in the router.

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

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

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

  You can add a maximum of 10 scripts simultaneously.

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

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

  Note
  Only SHA256 checksum is supported.

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

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

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

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

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

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

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

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

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

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

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

Example:

```
Router#show script status
Wed Aug 25 23:10:50.453 UTC
=============================================================================================
 Name              | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
 exec-script.py    | exec     | Config Checksum  | NEW         | Tue Aug 24 10:18:23 2021
 =============================================================================================
```

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

Configure Checksum for Exec Script

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

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

Note Exec scripts support SHA256 checksum.

Before you begin

Ensure that the script is added to the script management repository. See Download the Script to the Router, on page 79.

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:

```
Server$sha256sum sample1.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b sample1.py
```

Make note of the checksum value.

Step 2 View the status of the script.

Example:

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

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

 sample1.py                      | exec   | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
-------------------------------------------------------------------------------------------------------------

 Script Name       : sample1.py
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================================
```

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

Step 3 Enter global configuration mode.

Example:

```
Router#configure
```

Step 4 Configure the checksum.

Example:

```
Router(config)#script exec sample1.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Tue Aug 24 10:23:10.546 UTC
Router(config)#end
```

Step 5 Verify the status of the script.

Example:

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

 Name                            | Type   | Status           | Last Action | Action Time
```

```
-----------------------------------------------------------------------------------------------------

 sample1.py                      | exec   | Ready            | NEW         | Fri Aug 20 05:03:41 2021
-----------------------------------------------------------------------------------------------------

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

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script fails. It is mandatory for the checksum values to match for the script to run.

Run the Exec Script

To run an exec script, use the **script run** command. After the script is run, a request ID is generated. Each script run is associated with a unique request ID.

Before you begin

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

1. Download the Script to the Router, on page 79
2. Configure Checksum for Exec Script, on page 80

Procedure

Run the exec script.

Example:

```
Router#script run sample1.py
Wed Aug 25 16:40:59.134 UTC
Script run scheduled: sample1.py. Request ID: 1629800603
Script sample1.py (exec) Execution complete: (Req. ID 1629800603) : Return Value: 0 (Executed)
```

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

Keyword	Description
arguments	Script command-line arguments. Syntax: Strings in single quotes. Escape double quotes inside string arguments (if any). For example: `Router#script run sample1.py arguments 'hello world' '-r' '-t' 'exec' '--sleep' '5' description "Sample exec script"`
background	Run script in background. By default, the script runs in the foreground. When a script is run in the background, the console is accessible only after the script run is complete.
description	Description about the script run. `Router#script run sample1.py arguments '-arg1' 'reload' '-arg2' 'all' 'description' "Script reloads the router"` When you provide both the argument and description ensure that the arguments are in single quote and description is in double quotes.
log-level	Script logging level. The default value is `INFO`. You can specifiy what information is to be logged. The log level can be set to one of these options—Critical, Debug, Error, Info, or Warning.
log-path	Location to store the script logs. The default log file location is in the script management repository `harddisk:/mirror/script-mgmt/logs`.
max-runtime	Maximum run-time of script can be set between 1–3600 seconds. The default value is 300.

The script run is complete.

View the Script Execution Details

View the status of the script execution.

Before you begin

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

1. Download the Script to the Router, on page 79
2. Configure Checksum for Exec Script, on page 80
3. Run the Exec Script, on page 82

Procedure

Step 1 View the status of the script execution.

Example:

```
Router#show script execution
Wed Aug 25 18:32:12.351 UTC
==========================================================================================================

 Req. ID   | Name (type)              | Start                    | Duration   | Return | Status
----------------------------------------------------------------------------------------------------------

 1629800603| sample1.py (exec)        | Wed Aug 25 16:40:59 2021 | 60.62s     | 0      | Executed
==========================================================================================================
```

You can view detailed or filtered data for every script run.

Step 2 Filter the script execution status to view the detailed output of a specific script run via request ID.

Example:

```
Router#show script execution request-id 1629800603 detail output
Wed Aug 25 18:37:12.920 UTC
==========================================================================================================

 Req. ID   | Name (type)                             | Start                    | Duration   |
Return | Status
----------------------------------------------------------------------------------------------------------

 1629800603| sample1.py (exec)                       | Wed Aug 25 16:40:59 2021 | 60.62s     | 0
    | Executed
----------------------------------------------------------------------------------------------------------

 Execution Details:
 ------------------
 Script Name  : sample1.py
 Log location : /harddisk:/mirror/script-mgmt/logs/sample1.py_exec_1629800603
 Arguments    :
 Run Options  : Logging level - INFO, Max. Runtime - 300s, Mode - Foreground
 Events:
 -------
 1.   Event        : New
      Time         : Wed Aug 25 16:40:59 2021
      Time Elapsed : 0.00s Seconds
      Description  : None
 2.   Event        : Started
      Time         : Wed Aug 25 16:40:59 2021
      Time Elapsed : 0.03s Seconds
      Description  : Script execution started. PID (20736)
 3.   Event        : Executed
      Time         : Wed Aug 25 16:42:00 2021
      Time Elapsed : 60.62s Seconds
      Description  : Script execution complete
----------------------------------------------------------------------------------------------------------

 Script Output:
 --------------
 Output File  : /harddisk:/mirror/script-mgmt/logs/sample1.py_exec_1629800603/stdout.log
 Content      :
==========================================================================================================
```

Keyword	Description
detail	Display detailed script execution history, errors, output and deleted scripts. `Router#show script execution detail [errors | output | show-del]`

Keyword	Description
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}]`

Manage Scripts

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

Delete Exec Script from the Router

Delete the script from the script management repository using the **script remove** command. This repository stores the downloaded scripts.

Procedure

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

Example:

```
Router#show script status
Wed Aug 25 23:10:50.453 UTC
=============================================================================================================
Name       | Type   | Status          | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------
sample1.py | exec   | Config Checksum | NEW         | Tue Aug 24 10:18:23 2021
sample2.py | exec   | Config Checksum | NEW         | Wed Aug 25 23:44:53 2021
sample3.py | config | Config Checksum | NEW         | Wed Aug 25 23:44:57 2021
```

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

Step 2 Delete the script.

Example:

```
Router#script remove exec sample2.py
Wed Aug 25 231:46:38.170 UTC
sample2.py has been deleted from the script repository
```

You can also delete multiple scripts simulataneoulsy.

Step 3 Verify the script is deleted from the subdirectory.

Example:

```
Router#show script status
Wed Aug 25 23:48:50.453 UTC
=============================================================================================================
Name       | Type   | Status          | Last Action | Action Time
-------------------------------------------------------------------------------------------------------------
sample1.py | exec   | Config Checksum | NEW         | Tue Aug 24 10:18:23 2021
sample3.py | config | Config Checksum | NEW         | Wed Aug 25 10:44:57 2021
```

The script is deleted from the script management repository.

Example: Exec Script to Verify Bundle Interfaces

In this example, you create a script to verify the bandwidth usage of bundle interfaces on the router, and check if it is beyond the defined limit. If usage is above the limit, the script generates a syslog indicating that the bandwidth is above the limit, and additional interfaces must be added to the bundle.

Before you begin

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

1. Create an exec script `verify_bundle.py`. Store the script on an HTTP server or copy the script to the harddisk of the router.

```
"""
Bundle interfaces bandwidth verification script

Verify bundle interfaces mpls packets per sec is below threshold.
If pkts/sec is greater than threshold then print syslog message
and add list of new interfaces to bundle

Arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  Bundle interface name
  -t THRESHOLD, --threshold THRESHOLD
                        Bandwidth threshold
  -m MEMBERS, --members MEMBERS
                        interfaces (coma separated) to add to bundle
"""
import re
import argparse
from iosxr.xrcli.xrcli_helper import XrcliHelper
from cisco.script_mgmt import xrlog

syslog = xrlog.getSysLogger('verify_bundle')
log = xrlog.getScriptLogger('verify_bundle')
```

```
def add_bundle_members(bundle_name, members):

    helper = XrcliHelper()
    bundle_pattern = re.compile('[A-Z,a-z, ]([0-9]+)')
    match = bundle_pattern.search(bundle_name)
    if match:
        bundle_id = match.group(1)
    else:
        raise Exception('Invalid bundle name')
    cfg = ''
    for member in members:

        cfg = cfg + 'interface %s \nbundle id %s mode active\nno shutdown\n' % \
              (member.strip(), bundle_id)

    log.info("Configs to be added : \n%s" % cfg)
    result = helper.xr_apply_config_string(cfg)
    if result['status'] == 'success':
        msg = "Configuring new bundle members successful"
        syslog.info(msg)
        log.info(msg)
    else:
        msg = "Configuring new bundle members failed"
        syslog.warning(msg)
        log.warning(msg)


def verify_bundle(script_args):

    helper = XrcliHelper()
    cmd = "show interfaces %s accounting rates" % script_args.name
    cmd_out = helper.xrcli_exec(cmd)
    if not cmd_out['status'] == 'success':
        raise Exception('Invalid bundle or error getting interface accounting rates')

    log.info('Command output : \n%s' % cmd_out['output'])
    rate_pattern = re.compile("MPLS +[0-9]+ +[0-9]+ +[0-9]+ +([0-9]+)")
    match = rate_pattern.search(cmd_out['output'])
    if match:
        pktspersec = int(match.group(1))
        if pktspersec > int(script_args.threshold):
            msg = 'Bundle %s bandwidth of %d pps is above threshold of %s pps' % \
                    (script_args.name, pktspersec, script_args.threshold)
            log.info(msg)
            syslog.info(msg)
            return False
        else:
            msg = 'Bundle %s bandwidth of %d pps is below threshold of %s pps' % \
                    (script_args.name, pktspersec, script_args.threshold)
            log.info(msg)
            return True


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description="Verify budle")
    parser.add_argument("-n", "--name",
                        help="Bundle interface name")
    parser.add_argument("-t", "--threshold",
                        help="Bandwidth threshold")
    parser.add_argument("-m", "--members",
                        help="interfaces (coma separated) to add to bundle")
```

```
args = parser.parse_args()
log.info('Script arguments :')
log.info(args)
if not verify_bundle(args):
    syslog.info("Adding new members (%s) to bundle interfaces %s" %
                (args.members, args.name))
    add_bundle_members(args.name, args.members.split(','))
```

2. Add the script from HTTP server or harddisk to the script management repository. See Download the Script to the Router, on page 79.
3. Configure the checksum to verify the authenticity and integrity of the script.

Procedure

Step 1 View the script status.

Example:

```
Router#show script status
Sat Sep 25 00:10:11.222 UTC
================================================================================
Name                | Type   | Status   | Last Action | Action Time
--------------------------------------------------------------------------------
verify_bundle.py    | exec   | Ready    | MODIFY      | Sat Sep 25 00:08:55 2021
================================================================================
```

The status indicates that the script is ready to be run.

Step 2 Run the script.

Example:

```
Router#script run verify_bundle.py arguments '--name' 'Bundle-Ether6432' '-t'
'400000' '-m' 'FourHundredGigE0/0/0/2
Sat Sep 25 00:11:14.183 UTC
Script run scheduled: verify_bundle.py. Request ID: 1632528674
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Script arguments :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Namespace(members='FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3', name='Bundle-Ether6432', threshold='400000')
[2021-09-25 00:11:14,735] INFO [verify_bundle]:: Command output :

-------------- show interfaces Bundle-Ether6432 accounting rates --------------
Bundle-Ether6432
                              Ingress                        Egress
  Protocol             Bits/sec         Pkts/sec     Bits/sec         Pkts/sec
  IPV4_UNICAST            22000               40            0                0
  MPLS                        0                0   1979249000           430742
  ARP                         0                0            0                0
  IPV6_ND                     0                0            0                0
  CLNS                     1000                1        26000                3



[2021-09-25 00:11:14,736] INFO [verify_bundle]:: Bundle Bundle-Ether6432 bandwidth
of 430742 pps is above threshold of 400000 pps
[2021-09-25 00:11:14,737] INFO [verify_bundle]:: Configs to be added :
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
no shutdown
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
```

```
no shutdown

[2021-09-25 00:11:18,254] INFO [verify_bundle]:: Configuring new bundle members successful
Script verify_bundle.py (exec) Execution complete: (Req. ID 1632528674) : Return Value: 0 (Executed)
```

Step 3 View the detailed output based on request ID. A request ID is generated for each script run.

Example:

```
Router#show script execution request-id 1632528674 detail output
Sat Sep 25 00:11:58.141 UTC
==================================================================================================
Req. ID   | Name (type)                | Start                    | Duration   | Return | Status
--------------------------------------------------------------------------------------------------
1632528674| verify_bundle.py (exec)    | Sat Sep 25 00:11:14 2021 | 4.06s      | 0      | Executed
--------------------------------------------------------------------------------------------------
Execution Details:
------------------
Script Name  : verify_bundle.py
Log location : /harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674
Arguments    : '--name', 'Bundle-Ether6432', '-t', '400000', '-m', 'FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3'
Run Options  : Logging level - INFO, Max. Runtime - 300s, Mode - Foreground
Events:
-------
1.   Event        : New
      Time         : Sat Sep 25 00:11:14 2021
      Time Elapsed : 0.00s Seconds
      Description  : None
2.   Event        : Started
      Time         : Sat Sep 25 00:11:14 2021
      Time Elapsed : 0.02s Seconds
      Description  : Script execution started. PID (29768)
3.   Event        : Executed
      Time         : Sat Sep 25 00:11:18 2021
      Time Elapsed : 4.06s Seconds
      Description  : Script execution complete
--------------------------------------------------------------------------------------------------------
Script Output:
--------------
Output File  : /harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674/stdout.log
Content      :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Script arguments :
[2021-09-25 00:11:14,579] INFO [verify_bundle]:: Namespace(members='FourHundredGigE0/0/0/2,
FourHundredGigE0/0/0/3',
name='Bundle-Ether6432', threshold='400000')
[2021-09-25 00:11:14,735] INFO [verify_bundle]:: Command output :

-------------- show interfaces Bundle-Ether6432 accounting rates --------------
Bundle-Ether6432
                              Ingress                        Egress
  Protocol             Bits/sec         Pkts/sec     Bits/sec         Pkts/sec
  IPV4_UNICAST            22000               40            0                0
  MPLS                        0                0   1979249000           430742
  ARP                         0                0            0                0
  IPV6_ND                     0                0            0                0
  CLNS                     1000                1        26000                3

[2021-09-25 00:11:14,736] INFO [verify_bundle]:: Bundle Bundle-Ether6432 bandwidth of 430742 pps is
 above threshold
of 400000 pps
[2021-09-25 00:11:14,737] INFO [verify_bundle]:: Configs to be added :
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
```

```
no shutdown
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
no shutdown

[2021-09-25 00:11:18,254] INFO [verify_bundle]:: Configuring new bundle members successful
================================================================================================================
```

Step 4 View the running configuration for the bundle interfaces.

Example:

```
Router#show running-config interface FourHundredGigE0/0/0/2
Sat Sep 25 00:12:30.765 UTC
interface FourHundredGigE0/0/0/2
bundle id 6432 mode active
!

Router#show running-config interface FourHundredGigE0/0/0/3
Sat Sep 25 00:12:38.659 UTC
interface FourHundredGigE0/0/0/3
bundle id 6432 mode active
!
```

Step 5 View the latest logs for more details about the script run. Here, the last 10 logs are displayed. The logs show that configuring new bundle members is successful.

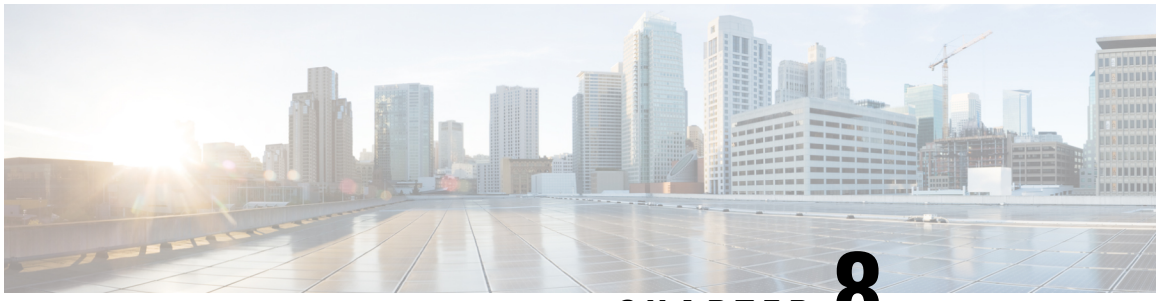
Example:

```
Router#show logging last 10
Sat Sep 25 00:13:34.383 UTC
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: level warnings, 178 messages logged
    Monitor logging: level debugging, 0 messages logged
    Trap logging: level informational, 0 messages logged
    Buffer logging: level debugging, 801 messages logged

Log Buffer (2097152 bytes):

RP/0/RP0/CPU0:Sep 25 00:10:05.763 UTC: config[66385]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'cisco'.
Use 'show configuration commit changes 1000000045' to view the changes.
RP/0/RP0/CPU0:Sep 25 00:10:07.971 UTC: config[66385]: %MGBL-SYS-5-CONFIG_I : Configured from console
 by cisco on vty0 (6.3.65.175)
RP/0/RP0/CPU0:Sep 25 00:11:14.447 UTC: script_control_cli[66627]: %OS-SCRIPT_MGMT-6-INFO :
Script-control: Script run scheduled:
verify_bundle.py. Request ID: 1632528674
RP/0/RP0/CPU0:Sep 25 00:11:14.453 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Script execution
verify_bundle.py (exec) Started : Request ID : 1632528674 :: PID: 29768
RP/0/RP0/CPU0:Sep 25 00:11:14.453 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Starting execution
verify_bundle.py (exec) (Req. ID: 1632528674) : Logs directory:
/harddisk:/mirror/script-mgmt/logs/verify_bundle.py_exec_1632528674
RP/0/RP0/CPU0:Sep 25 00:11:14.736 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
 Bundle Bundle-Ether6432
bandwidth of 430742 pps is above threshold of 400000 pps
RP/0/RP0/CPU0:Sep 25 00:11:14.736 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
 Adding new members
(FourHundredGigE0/0/0/2, FourHundredGigE0/0/0/3) to bundle interfaces Bundle-Ether6432
RP/0/RP0/CPU0:Sep 25 00:11:16.916 UTC: config[66655]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'cisco'. Use 'show
configuration commit changes 1000000046' to view the changes.
RP/0/RP0/CPU0:Sep 25 00:11:18.254 UTC: python3_xr[66632]: %OS-SCRIPT_MGMT-6-INFO : Script-verify_bundle:
```

```
 Configuring new bundle members
successful
RP/0/RP0/CPU0:Sep 25 00:11:18.497 UTC: script_agent_main[347]: %OS-SCRIPT_MGMT-6-INFO :
Script-script_agent: Script verify_bundle.py
(exec) Execution complete: (Req. ID 1632528674) : Return Value: 0 (Executed)
```



CHAPTER 8

Process Scripts

Cisco IOS XR process scripts are also called daemon scripts. The process scripts are persistent scripts that continue to run as long as you have activated the scripts. An IOS XR process, Application manager (AppMgr or app manager), manages the lifecycle of process scripts. The scripts are registered as an application on the app manager. This application represents the instance of the script that is running on the router.

The app manager is used to:

- Start, stop, monitor, or retrieve the operational status of the script.
- Maintain the startup dependencies between the processes.
- Restart the process if the script terminates unexpectedly based on the configured restart policy.

Process scripts support Python 3.5 programming language. For the list of supported packages, see Cisco IOS XR Python Packages.

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

Note This chapter does not delve into creating Python scripts, but assumes that you have basic understanding of Python programming language. This section will walk you through the process involved in deploying and using the scripts on the router. A process script refers to code that runs continuously or endlessly.

- Workflow to Run Process Scripts, on page 93
- Managing Actions on Process Script, on page 103
- Example: Check CPU Utilization at Regular Intervals Using Process Script, on page 103

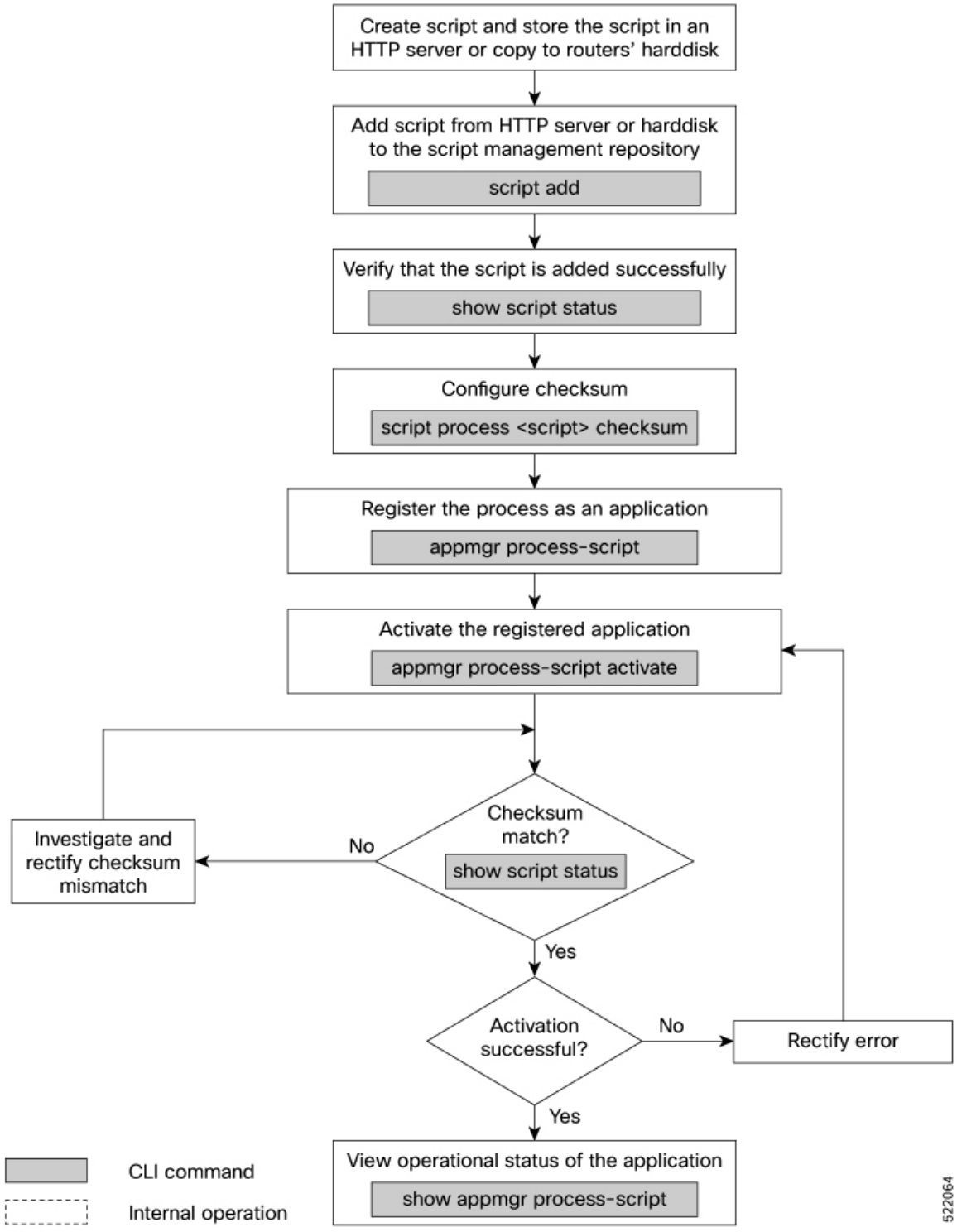
Workflow to Run Process Scripts

Complete the following tasks to provision process scripts:

- Download the script—Store the script on an external server or copy to the harddisk of the router. Add the script from the external server or harddisk to the script management repository on the router using the **script add process** command.
- Configure the checksum—Check script integrity and authenticity using the **script process <script.py> checksum** command.

- Register the script—Register the script as an application in the app manager using **appmgr process-script** command.
- Activate the script—Activate the registered application using **appmgr process-script activate** command.
- View the script execution details—Retrieve the operational data using the **show appmgr process-script** command.

The following image shows the workflow diagram representing the steps that are involved in using a process script:


Create script and store the script in an HTTP server or copy to routers' harddisk
Add script from HTTP server or harddisk to the script management repository
script add
Verify that the script is added successfully
show script status
Configure checksum
script process <script> checksum
Register the process as an application
appmgr process-script
Activate the registered application
appmgr process-script activate
Checksum match?
show script status
No
Investigate and rectify checksum mismatch
Yes
Activation successful?
No
Rectify error
Yes
View operational status of the application
show appmgr process-script
CLI command
Internal operation
522064

Download the Script to the Router

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

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

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

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

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

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 `process-script.py` downloaded from an external repository `http://192.0.2.0/scripts`:

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

  The `script add process` supports the HTTP, HTTPS, FTP, TFTP, and SCP protocols for copying a script.

  You can add a maximum of 10 scripts simultaneously.

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

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

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

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

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

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

Note
Only SHA256 checksum is supported.

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

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

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

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

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

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

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

Example:

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

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

Configure Checksum for Process Script

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

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

Note Process scripts support the SHA256 checksum hash.

Before you begin

Ensure that the script is added to the script management repository. See Download the Script to the Router, on page 79.

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
Fri Aug 20 05:04:13.539 UTC
=============================================================================================
 Name                    | Type     | Status           | Last Action | Action Time
---------------------------------------------------------------------------------------------
process-script.py        | process  | Config Checksum  | NEW         | Fri Aug 20 05:03:41 2021
---------------------------------------------------------------------------------------------
 Script Name       : process-script.py
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:03:41 2021
      Description  : User action IN_CLOSE_WRITE
=============================================================================================
```

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

Step 3 Configure the checksum.

Example:

```
Router#configure
Router(config)#script process process-script.py checksum SHA256
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
Router(config)#commit
Tue Aug 20 05:10:10.546 UTC
Router(config)#end
```

Step 4 Verify the status of the script.

Example:

```
Router#show script status detail
Fri Aug 20 05:15:17.296 UTC
=============================================================================================
 Name                  | Type     | Status      | Last Action | Action Time
---------------------------------------------------------------------------------------------
 process-script.py     | process  | Ready       | NEW         | Fri Aug 20 05:20:41 2021
---------------------------------------------------------------------------------------------
 Script Name       : process-script.py
 Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
 History:
 --------
 1.   Action       : NEW
      Time         : Fri Aug 20 05:20:41 2021
      Checksum     : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
```

```
      Description  : User action IN_CLOSE_WRITE
==============================================================================================
```

The status `Ready` indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script fails. It is mandatory for the checksum values to match for the script to run.

Register the Process Script as an Application

Register the process script with the app manager to enable the script. The registration is mandatory for using process script on the router.

Before you begin

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

- Download the Script to the Router, on page 79
- Configure Checksum for Process Script, on page 97

Procedure

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

Example:

```
Router#configure
Fri Aug 20 06:10:19.284 UTC
Router(config)#appmgr process-script my-process-app
Router(config-process)#executable process-script.py
```

Here, `my-process-app` is the application for the executable `process-script.py` script.

Step 2 Provide the arguments for the script.

Example:

```
Router(config-process)#run-args --host <host-name> --runtime 3 --log script
```

Step 3 Set a restart policy for the script if there is an error.

Example:

```
Router(config-process)#restart on-failure max-retries 3
Router(config-process)#commit
```

Here, the maximum attempts to restart the script is set to 3. After 3 attempts, the script stops.

You can set more options to restart the process:

Keyword	Description
always	Always restart automatically. If the process exits, a scheduler queues the script and restarts the script. **Note** This is the default restart policy.
never	Never restart automatically. If the process exits, the script is not rerun unless you provide an action command to invoke the process.
on-failure	Restart on failure automatically. If the script exits successfully, the script is not scheduled again.
unless-errored	Restart script automatically unless errored.
unless-stopped	Restart script automatically unless stopped by the user using an action command.

Step 4 View the status of the registered script.

Example:

```
Router#show appmgr process-script-table
Fri Aug 20 06:15:44.244 UTC
Name            Executable          Activated    Status      Restart Policy   Config Pending
--------------- ------------------- --------- ------------- ---------------- --------------
my-process-app  process-script.py      No      Not Started  On Failure             No
```

The script is registered but is not active.

Activate the Process Script

Activate the process script that you registered with the app manager.

Before you begin

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

- Download the Script to the Router, on page 79
- Configure Checksum for Process Script, on page 97
- Register the Process Script as an Application, on page 99

Procedure

Step 1 Activate the process script.

Example:

```
Router#appmgr process-script activate name my-process-app
Fri Aug 20 06:20:55.006 UTC
```

The instance `my-process-app` is activated for the process script.

Step 2 View the status of the activated script.

Example:

```
Router#show appmgr process-script-table
Fri Aug 20 06:22:03.201 UTC
Name             Executable         Activated    Status     Restart Policy   Config Pending
---------------  ------------------ ---------- ------------- ---------------- --------------
my-process-app   process-script.py      Yes       Running    On Failure            No
```

The process script is activated and running.

Note
You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes are pending. The status of the modification is indicated in the `Config Pending` option. In the example, value `No` indicates that there are no configuration changes that must be activated.

Obtain Operational Data and Logs

Retrieve the operational data and logs of the script.

Before you begin

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

- Download the Script to the Router, on page 79
- Configure Checksum for Process Script, on page 97
- Register the Process Script as an Application, on page 99
- Activate the Process Script, on page 100

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
Fri Aug 20 06:20:21.947 UTC
Application: my-process-app

  Registration info:
    Executable                  : process-script.py
    Run arguments               : --host <host-name> --runtime 3 --log script
    Restart policy              : On Failure
    Maximum restarts            : 3

  Pending Configuration:
    Run arguments               : --host <host-name> --runtime 3 --log script
    Restart policy              : Always
```

```
  Execution info and status:
    Activated                     : Yes
    Status                        : Running
    Executable Checksum          : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b

    Last started time             : Fri Aug 20 06:20:21.947
    Restarts since last activate  : 0/3
    Log location                  :
/harddisk:/mirror/script-mgmt/logs/process-script.py_process_my-process-app
    Last exit code                : 1
```

Step 2 View the logs for the process scripts. App manager shows the logs for errors and output.

Example:

The following example shows the output logs:

```
Router#show appmgr process-script my-process-app logs output
Fri Aug 20 06:25:20.912 UTC
[2021-08-20 06:20:55,609] INFO [sample-process]:: Beginning execution of process..
[2021-08-20 06:20:55,609] INFO [sample-process]:: Connecting to host '<host-name>'
[2021-08-20 06:20:56,610] INFO [sample-process]:: Reading database..
[2021-08-20 06:20:58,609] INFO [sample-process]:: Listening for requests..
```

The following example shows the error logs with errors:

```
Router#show appmgr process-script my-process-app logs errors
Fri Aug 20 06:30:20.912 UTC
----------Run ID:1632914459  Fri Aug 20 06:30:20 2021----------
Traceback (most recent call last):
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 121, in <module>
    main(args)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 97, in main
    printer()
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 37, in wrapper
    result = func(*args, **kwargs)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 88, in printer
    time.sleep(1)
  File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 30, in _handle_timeout
    raise TimeoutError(error_message)
__main__.TimeoutError: Timer expired
----------Run ID:1632914460  Fri Aug 20 06:31:03 2021----------
```

This example shows the log without errors:

```
Router#show appmgr process-script my-process-app logs errors
Fri Aug 20 06:30:20.912 UTC
----------Run ID:1624346220  Fri Aug 20 10:46:44 2021----------
----------Run ID:1624346221  Fri Aug 20 10:47:50 2021----------
----------Run ID:1624346222  Fri Aug 20 10:52:39 2021----------
----------Run ID:1624346223  Fri Aug 20 10:53:45 2021----------
----------Run ID:1624346224  Fri Aug 20 11:07:17 2021----------
----------Run ID:1624346225  Fri Aug 20 11:08:23 2021----------
----------Run ID:1624346226  Fri Aug 20 11:09:29 2021----------
----------Run ID:1624346227  Fri Aug 20 11:10:35 2021----------
----------Run ID:1624346228  Fri Aug 20 11:11:41 2021----------
```

Managing Actions on Process Script

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

Table 7: Feature History Table

Action	Description
Deactivate	Clears all the resources that the application uses. `Router#appmgr process-script deactivate name my-process-app` You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes do not take effect.
Kill	Terminates the script if the option to stop the script is unresponsive. `Router#appmgr process-script kill name my-process-app`
Restart	Restarts the process script. `Router#appmgr process-script restart name my-process-app`
Start	Starts an application that is already registered and activated with the app manager. `Router#appmgr process-script start name my-process-app`
Stop	Stops an application that is already registered, activated, and is currently running. Only the application is stopped; resources that the application uses is not cleared. `Router#appmgr process-script stop name my-process-app`

Example: Check CPU Utilization at Regular Intervals Using Process Script

In this example, you use the process script to check CPU utilization at regular intervals. The script does the following actions:

- Monitor the CPU threshold value.
- If the threshold value equals or exceeds the value passed as argument to the script, log an error message that the threshold value has exceeded.

Before you begin

Ensure you have completed the following prerequisites before you register and activate the script:

1. Create a process script `cpu-utilization-process.py`. Store the script on an external server or copy the script to the harddisk of the router.

```
import time
import os
```

```
import xmltodict
import re
import argparse

from cisco.script_mgmt import xrlog
from iosxr.netconf.netconf_lib import NetconfClient

log = xrlog.getScriptLogger('Sample')
syslog = xrlog.getSysLogger('Sample')

def cpu_memory_check(threshold):
    """
    Check total routes in router
    """
    filter_string = """
    <system-monitoring xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-wdsysmon-fd-oper">
      <cpu-utilization>
        <node-name>0/RP0/CPU0</node-name>
          <total-cpu-one-minute/>
      </cpu-utilization>
    </system-monitoring>"""
    nc = NetconfClient(debug=True)
    nc.connect()
    do_get(nc, filter=filter_string)
    ret_dict = _xml_to_dict(nc.reply, 'system-monitoring')
    total_cpu =
int(ret_dict['system-monitoring']['cpu-utilization']['total-cpu-one-minute'])
    if total_cpu >= threshold:
        syslog.error("CPU utilization is %s, threshold value is %s"
%(str(total_cpu),str(threshold)))
    nc.close()

def _xml_to_dict(xml_output, xml_tag=None):
    """
    convert netconf rpc request to dict
    :param xml_output:
    :return:
    """
    if xml_tag:
        pattern = "<data>\s+(<%s.*</%s>).*</data>" % (xml_tag, xml_tag)
    else:
        pattern = "(<data>.*</data>)"
    xml_output = xml_output.replace('\n', ' ')
    xml_data_match = re.search(pattern, xml_output)
    ret_dict = xmltodict.parse(xml_data_match.group(1))
    return ret_dict

def do_get(nc, filter=None, path=None):
    try:
        if path is not None:
            nc.rpc.get(file=path)
        elif filter is not None:
            nc.rpc.get(request=filter)
        else:
            return False
    except Exception as e:
        return False
    return True

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("threshold", help="cpu utilization threshold",type=int)
    args = parser.parse_args()
    threshold = args.threshold
```

```
while(1):
    cpu_memory_check(threshold)
    time.sleep(30)
```

Configure the script with the desired threshold criteria. This default threshold is configured to alert when CPU utilization exceeds this value. The script checks the CPU utilization every 30 seconds.

2. Add the script from the external server or harddisk to the script management repository. See Download the Script to the Router, on page 79.
3. Configure the checksum to verify the authenticity and integrity of the script. See Configure Checksum for Process Script, on page 97.

Procedure

Step 1 Register the process script `cpu-utilization-process.py` with an instance name `my-process-app` in the app manager.

Example:

```
Router(config)#appmgr process-script my-process-app
Router(config-process)#executable cpu-utilization-process.py
Router(config-process)#run-args <threshold-value>
```

Step 2 Activate the registered application.

Example:

```
Router(config-process)#appmgr process-script activate name my-process-app
```

Step 3 Check the script status.

Example:

```
Router#show appmgr process-script-table
Thu Sep 30 18:15:03.201 UTC
Name             Executable                  Activated    Status     Restart Policy   Config Pending
---------------  --------------------------  ---------  ------------ ---------------- --------------
my-process-app   cpu-utilization-process.py      Yes       Running    On Failure            No
```

Step 4 View the log.

Example:

```
Router#show appmgr process-script my-process-app logs errors
RP/0/RP0/CPU0:Sep 30 18:03:54.391 UTC: python3_xr[68378]: %OS-SCRIPT_MGMT-3-ERROR :
Script-test_process: CPU utilization is 6, threshold value is 5
```

An error message is displayed that the CPU utilization has exceeded the configured threshold value, and helps you take corrective actions.