Cisco Nexus 7000 Series NX-OS Programmability Guide

First Published: 2016-12-23

Last Modified: 2020-02-10

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

CONTENTS

Preface

The preface contains the following sections:

Audience

This publication is for network administrators who configure and maintain Cisco Nexus devices.

Document Conventions

Note As part of our constant endeavor to remodel our documents to meet our customers' requirements, we have modified the manner in which we document configuration tasks. As a result of this, you may find a deviation in the style used to describe these tasks, with the newly included sections of the document following the new format.

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
Italic	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).
[x | y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.

Convention	Description
{x | y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y | z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.
`variable`	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

Convention	Description
`screen font`	Terminal sessions and information the switch displays are in screen font.
`boldface screen font`	Information you must enter is in boldface screen font.
italic screen font	Arguments for which you supply values are in italic screen font.
< >	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

This document uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation for Cisco Nexus 7000 Series NX-OS Software

The entire Cisco Nexus 7000 Series NX-OS documentation set is available at the following URL:

https://www.cisco.com/c/en/us/support/switches/nexus-7000-series-switches/series.html#~tab-documents

Release Notes

The release notes are available at the following URL:

http://www.cisco.com/en/US/products/ps9402/prod_release_notes_list.html

Configuration Guides

These guides are available at the following URL:

http://www.cisco.com/en/US/products/ps9402/products_installation_and_configuration_guides_list.html

The documents in this category include:

- *Cisco Nexus 7000 Series NX-OS Configuration Examples*
- *Cisco Nexus 7000 Series NX-OS FabricPath Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Fundamentals Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Interfaces Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS IP SLAs Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Layer 2 Switching Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS LISP Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS MPLS Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Multicast Routing Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS OTV Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Quality of Service Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS SAN Switching Guide*
- *Cisco Nexus 7000 Series NX-OS Security Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS System Management Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Unicast Routing Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Verified Scalability Guide*
- *Cisco Nexus 7000 Series NX-OS Virtual Device Context Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Virtual Device Context Quick Start*
- *Cisco Nexus 7000 Series NX-OS OTV Quick Start Guide*
- *Cisco NX-OS FCoE Configuration Guide for Cisco Nexus 7000 and Cisco MDS 9500*
- *Cisco Nexus 2000 Series Fabric Extender Software Configuration Guide*

Command References

These guides are available at the following URL:

http://www.cisco.com/en/US/products/ps9402/prod_command_reference_list.html

The documents in this category include:

- *Cisco Nexus 7000 Series NX-OS Command Reference Master Index*
- *Cisco Nexus 7000 Series NX-OS FabricPath Command Reference*
- *Cisco Nexus 7000 Series NX-OS Fundamentals Command Reference*
- *Cisco Nexus 7000 Series NX-OS High Availability Command Reference*
- *Cisco Nexus 7000 Series NX-OS Interfaces Command Reference*
- *Cisco Nexus 7000 Series NX-OS Layer 2 Switching Command Reference*
- *Cisco Nexus 7000 Series NX-OS LISP Command Reference*
- *Cisco Nexus 7000 Series NX-OS MPLS Configuration Guide*
- *Cisco Nexus 7000 Series NX-OS Multicast Routing Command Reference*
- *Cisco Nexus 7000 Series NX-OS OTV Command Reference*
- *Cisco Nexus 7000 Series NX-OS Quality of Service Command Reference*
- *Cisco Nexus 7000 Series NX-OS SAN Switching Command Reference*
- *Cisco Nexus 7000 Series NX-OS Security Command Reference*
- *Cisco Nexus 7000 Series NX-OS System Management Command Reference*
- *Cisco Nexus 7000 Series NX-OS Unicast Routing Command Reference*
- *Cisco Nexus 7000 Series NX-OS Virtual Device Context Command Reference*
- *Cisco NX-OS FCoE Command Reference for Cisco Nexus 7000 and Cisco MDS 9500*

Other Software Documents

You can locate these documents starting at the following landing page:

https://www.cisco.com/c/en/us/support/switches/nexus-7000-series-switches/series.html#~tab-documents

- *Cisco Nexus 7000 Series NX-OS MIB Quick Reference*
- *Cisco Nexus 7000 Series NX-OS Software Upgrade and Downgrade Guide*
- *Cisco Nexus 7000 Series NX-OS Troubleshooting Guide*
- *Cisco NX-OS Licensing Guide*
- *Cisco NX-OS System Messages Reference*
- *Cisco NX-OS Interface User Guide*

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to: .

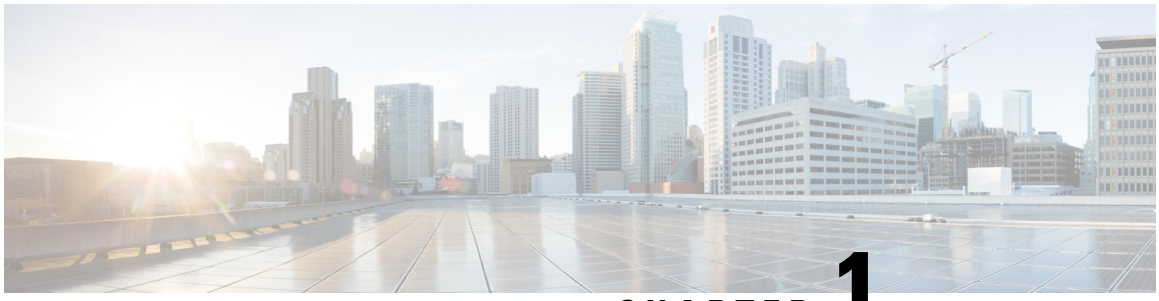
We appreciate your feedback.

Communications, Services, and Additional Information

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

Cisco Bug Search Tool

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



CHAPTER 1

New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 7000 Series NX-OS Programmability Guide*.

• New and Changed Information, on page 1

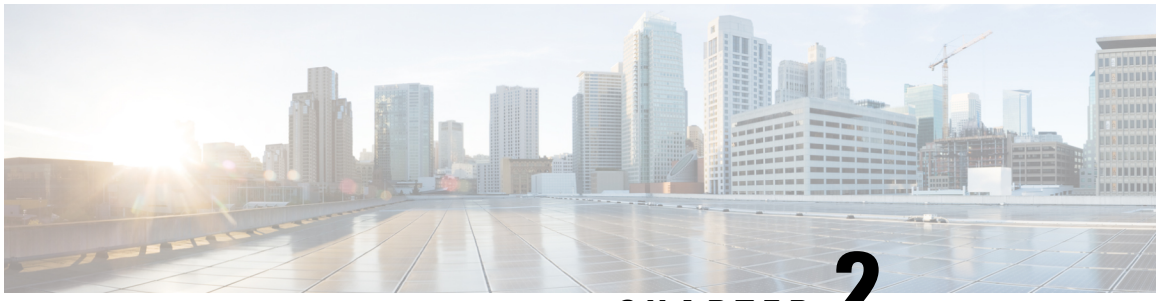
New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 7000 Series NX-OS Programmability Guide* and provides details about where they are documented.

Table 1: New and Changed Features

Feature Name	Description	Changed in Release	Where Documented
NX-API	The **nxapi use-vrf** feature is introduced, which helps to secure NX-API by binding the NX-API to a particular VRF.	8.2(3)	NX-API, on page 9
Streaminng Telemetry	Telemetry enables a push model for continuously streaming data from the show commands that are XMLized.	8.3(1)	Model-Driven Telemetry
NX-API	The following enhancements have been made to NX-API sandbox in this release: • Configuration Validation • Configuration Lock • Checkpoint Rollback • Command Reference • Generation of Java and JavaScript code that can be used for automation	8.0(1)	NX-API, on page 9

Feature Name	Description	Changed in Release	Where Documented
Python API	To use the Python CLI command APIs such as cli(), clid(), and clip() in Cisco NX-OS Release 8.0(1), you must first enable the API using the **from cli import *** command.	8.0(1)	Python API, on page 23



CHAPTER 2

Overview

Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 7000 Series devices is as follows:

- **Resilient**

  Provides critical business-class availability.

- **Modular**

  Has extensions that accommodate business needs.

- **Highly Programmatic**

  Allows for rapid automation and orchestration through Application Programming Interfaces (APIs).

- **Secure**

  Protects and preserves data and operations.

- **Flexible**

  Integrates and enables new technologies.

- **Scalable**

  Accommodates and grows with the business and its requirements.

- **Easy to use**

  Reduces the amount of learning required, simplifies deployment, and provides ease of manageability.

With the Cisco NX-OS operating system, the device functions in the unified fabric mode to provide network connectivity with programmatic automation functions.

Cisco NX-OS contains Open Source Software (OSS) and commercial technologies that provide automation, orchestration, programmability, monitoring and compliance support.

Standard Network Manageability Features

- SNMP (V1, V2, V3)
- Syslog
- RMON
- NETCONF
- CLI and CLI scripting

Advanced Automation Feature

The enhanced Cisco NX-OS on the device supports automation. The platform includes support for PowerOn Auto Provisioning (POAP).

The enhanced Cisco NX-OS on the device supports automation. The platform includes the following features that support automation:

- PowerOn Auto Provisioning (POAP) support
- XMPP support
- Chef and Puppet integration
- OpenStack integration
- OpenDayLight integration and OpenFlow support

PowerOn Auto Provisioning Support

PowerOn Auto Provisioning (POAP) automates the process of installing/upgrading software images and installing configuration files on Cisco Nexus devices that are being deployed in the network for the first time. It reduces the manual tasks required to scale the network capacity.

When a Cisco Nexus device with the POAP feature boots and does not find the startup configuration, the device enters POAP mode. It locates a DHCP server and bootstraps itself with its interface IP address, gateway, and DNS server IP addresses. The device obtains the IP address of a TFTP server or the URL of an HTTP server and downloads a configuration script that enables the device to download and install the appropriate software image and configuration file.

For more details about POAP, see the *Cisco Nexus 7000 Series NX-OS Fundamentals Configuration Guide*.

OpenStack Integration

The Cisco Nexus 7000 Series devices support the Cisco Nexus plugin for OpenStack Networking, also known as Neutron (http://www.cisco.com/web/solutions/openstack/index.html). The plugin allows you to build an infrastructure as a service (IaaS) network and to deploy a cloud network. With OpenStack, you can build an on-demand, self-service, multitenant computing infrastructure. However, implementing OpenStack's VLAN networking model across virtual and physical infrastructures can be difficult.

The OpenStack Networking extensible architecture supports plugins to configure networks directly. However, when you choose a network plugin, only that plugin's target technology is configured. When you are running OpenStack clusters across multiple hosts with VLANs, a typical plugin configures either the virtual network infrastructure or the physical network, but not both.

The Cisco Nexus plugin solves this difficult problem by including support for configuring both the physical and virtual networking infrastructure.

The Cisco Nexus plugin accepts OpenStack Networking API calls and uses the Network Configuration Protocol (NETCONF) to configure Cisco Nexus devices as well as Open vSwitch (OVS) that runs on the hypervisor. The Cisco Nexus plugin configures VLANs on both the physical and virtual network. It also allocates scarce VLAN IDs by deprovisioning them when they are no longer needed and reassigning them to new tenants whenever possible. VLANs are configured so that virtual machines that run on different virtualization (compute) hosts that belong to the same tenant network transparently communicate through the physical network. In addition, connectivity from the compute hosts to the physical network is trunked to allow traffic only from the VLANs that are configured on the host by the virtual switch.

The following table lists the features of the Cisco Nexus plugin for OpenStack Networking:

Table 2: Summary of Cisco Nexus Plugin features for OpenStack Networking (Neutron)

Considerations	Description	Cisco Nexus Plugin
Extension of tenant VLANs across virtualization hosts	VLANs must be configured on both physical and virtual networks. OpenStack Networking supports only a single plugin at a time. You must choose which parts of the networks to manually configure.	Accepts networking API calls and configures both physical and virtual switches.
Efficient use of scarce VLAN IDs	Static provisioning of VLAN IDs on every switch rapidly consumes all available VLAN IDs, which limits scalability and makes the network vulnerable to broadcast storms.	Efficiently uses limited VLAN IDs by provisioning and deprovisioning VLANs across switches as tenant networks are created and destroyed.
Easy configuration of tenant VLANs in a top-of-rack (ToR) switch	You must statically provision all available VLANs on all physical switches. This process is manual and error prone.	Dynamically provisions tenant-network-specific VLANs on switch ports connected to virtualization hosts through the Nexus plugin driver.
Intelligent assignment of VLAN IDs	Switch ports connected to virtualization hosts are configured to handle all VLANs. Hardware limits are reached quickly.	Configures switch ports connected to virtualization hosts only for the VLANs that correspond to the networks configured on the host. This feature enables accurate port and VLAN associations.

Considerations	Description	Cisco Nexus Plugin
Aggregation switch VLAN configuration for large multirack deployments.	When compute hosts run in several racks, you must fully mesh top-of-rack switches or manually trunk aggregation switches.	Supports Cisco Nexus 2000 Series Fabric Extenders to enable large, multirack deployments and eliminates the need for an aggregation switch VLAN configuration.

Integration Between Ansible and the Nexus Platform

Ansible is an open source IT configuration management and automation tool. Similar to Puppet and Chef, Ansible enable adminstrators to manage, automate, and orchestrate various types of server environments. Unlike Puppet and Chef, Ansible is agentless, and does not require a software agent to be installed on the target node (server or switch) in order to automate the device. By default, Ansible requires SSH and Python support on the target node, but Ansible can also be easily extended to use any API. Ansible modules make API calls against the NX-API to gather real-time state data and to make configuration changes on Cisco Nexus devices. For more details about Ansible, see Ansible's official documentation.

NX-API is a REST-like API for NX-OS based systems. NX-API allows network administrators and programmers to send CLI commands in an API call down to a network device eliminating the need for expect scripting since nearly all communication for NX-API uses structured data. Admins can choose from JSON, XML, or JSON-RPC. Support for NX-API is available on the Nexus 7000 series switches. In configuration mode on NX-OS platform, enter the **feature ?** command to see if nxapi is listed as an available feature.

Support for Ansible integration is available on the Nexus 7000 series switches. Integration between Ansible and the CiscoNexus 7000 is accomplished by using Ansible's open and extensible framework along with NX-API.

Programmability Support

Cisco NX-OS on Cisco Nexus 7000 Series devices support the following capabilities to aid programmability:

- NX-API support
- Python scripting
- Tcl scripting

NX-API Support

Cisco NX-API allows for HTTP-based programmatic access to the Cisco Nexus 7000 Series platform. This support is delivered by NX-API, an open source webserver. NX-API provides the configuration and management capabilities of the Cisco NX-OS CLI with web-based APIs. The device can be set to publish the output of the API calls in XML or JSON format. This API enables rapid development on the Cisco Nexus 7000 Series platform.

Python Scripting

Cisco Nexus 7000 Series devices support Python v2.7.2 in both interactive and non-interactive (script) modes.

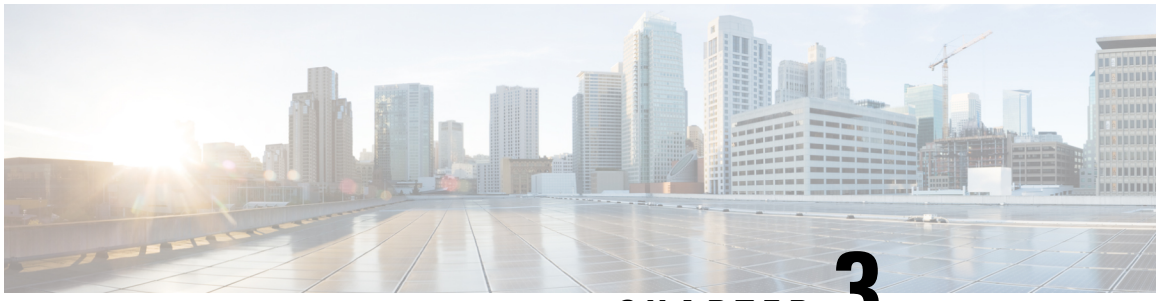
The Python scripting capability on the devices provide programmatic access to the switch CLI to perform various tasks, and to Power-On Auto Provisioning (POAP) and Embedded Event Manager (EEM) actions. Responses to Python calls that invoke the Cisco NX-OS CLI return text or JSON output.

The Python interpreter is included in the Cisco NX-OS software.

For more details about the Cisco Nexus 7000 Series devices support for Python, see the *Cisco Nexus 7000 Series NX-OS Fundamentals Configuration Guide*.

Tcl Scripting

Cisco Nexus 7000 Series devices support tcl (Tool Command Language). Tcl is a scripting language that enables greater flexibility with CLI commands on the switch. You can use tcl to extract certain values in the output of a **show** command, perform switch configurations, run Cisco NX-OS commands in a loop, or define EEM policies in a script.



CHAPTER 3

NX-API

This chapter contains the following sections:

About NX-API

In Cisco Nexus devices, CLIs are run only on the device. NX-API improves the accessibility of these CLIs by making them available outside the switch by using HTTP or HTTPS. You can use either of these extensions to the existing Cisco Nexus CLI system on the Cisco Nexus 7000 Series devices. NX-API supports **show** commands and configurations.

NX-API supports JSON-RPC, JSON, and XML formats.

NX-API generates a new certificate on each device when it communicates with them. This new/auto-generated certificate is valid only for 24 hours. NX-API does not use a default or hard-coded or an outdated certificate for its communication.

NX-API comes up with a default self-signed certificate and as mentioned earlier it is valid only for 24 hours. If you need to auto renew the certificate you must re-enable the **feature nxapi** command. You need to use your own certficate without depending on the NX-API certificate. Some browsers might treat this certificate as invalid. In that case you need to add an exception for the same in your settings and continue with your tasks.

Transport

NX-API uses HTTP or HTTPS as its transport. CLIs are encoded into the HTTP POST or HTTPS POST body.

The NX-API backend uses the Nginx HTTP server.

Message Format

NX-API is an enhancement to the Cisco Nexus 7000 Series CLI system, which supports XML output.

Note

- NX-API XML output presents information in a user-friendly format.
- NX-API XML does not map directly to the Cisco NX-OS NETCONF implementation.
- NX-API XML output can be converted into JSON.

Security

NX-API supports HTTP and HTTPS. If you use HTTPs, all communication to the device is encrypted.

NX-API is integrated into the authentication system on the device. Users must have appropriate accounts to access the device through NX-API, which uses HTTP basic authentication. All requests must contain the username and password in the HTTP header.

Note We recommend that you consider using HTTPS to secure your users' login credentials.

You can enable NX-API by using the **feature manager** command.

Prior to Cisco NX-OS Release 8.2(3) NX-API was accessible on all Layer 3 interfaces and there was no way to restrict the access to a particular VRF. The **nxapi use-vrf** feature is introduced in Cisco NX-OS Release 8.2(3), which helps to secure NX-API by binding the NX-API to a particular VRF.

You can Use ACLs to restrict HTTP or HTTPS access to a device along with VRF, if you want to restrict Nx-API access particular ip. For information about configuring ACLs, see the Cisco Nexus 7000 Series NX-OS Security Configuration Guide.

NX-API provides a session-based cookie, nxapi_auth, when users successfully authenticate for the first time. Along with the session cookie, the username and password are included in all subsequent NX-API requests that are sent to the device. The username and password are used with the session cookie to bypass the task of performing the entire authentication process again. If the session cookie is not included with subsequent requests, another session cookie is required and is provided by the authentication process. Avoiding multiple authentications helps reduce the devices' workload.

Note

- An nxapi_auth cookie expires in 600 seconds (10 minutes). This value is a fixed and cannot be adjusted.
- NX-API performs authentication through a programmable authentication module (PAM) on the switch. Use cookies to reduce the number of PAM authentications, which reduces the load on the PAM.

Using NX-API

Note For information about NX-API response codes, see NX-API Response Codes, on page 81.

By default, NX-API is disabled. Enable NX-API with the **feature manager** CLI command on the device.

The following example shows how to configure and launch the NX-API sandbox:

1. Enable the management interface:

```
switch# configure terminal
switch(config)# interface mgmt 0
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

2. Enable the NX-API **nxapi** feature:

```
switch# configure terminal
switch(config)# feature nxapi
```

Example

The following example shows a request and its response in XML format:

Request

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>cli_show</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
      <hostname>switch</hostname>
     </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

The following example shows a request and its response in JSON format:

Request:

```
{
  "ins_api": {
    "version": "1.0",
    "type": "cli_show",
    "chunk": "0",
    "sid": "1",
    "input": "show switchname",
```

```
    "output_format": "json"
  }
}
```

Response

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "1.0",
    "sid": "eoc",
    "outputs": {
      "output": {
        "input": "show switchname",
        "msg": "Success",
        "code": "200",
        "body": {
          "hostname": "switch"
        }
      }
    }
  }
}
```

The following example shows a request and response in JSON-RPC format:

Request

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show switchname",
      "version": 1
    },
    "id": 1
  }
]
```

Response

```
{
  "jsonrpc": "2.0",
  "result": {
    "body": {
      "hostname": "switch"
    }
  },
  "id": 1
}
```

Sending Requests

To send an NX-API request via HTTP, use *http://<ip-address-of-switch>/ins*.

To send an NX-API request with additional security via HTTPS, use *https://<ip-address-of-switch>/ins*.

The IP address of the management interface is *ip-address-of-switch>*.

Note The HTTP request must contain the content-type field in the header. For JSON-RPC requests, this must be application/json-rpc. For proprietary formats, this should be either text/xml or text/json, depending on the input format being used.

Obtaining XSD Files

Step 1 From your browser, navigate to the Cisco software download site:

http://software.cisco.com/download/navigator.html

The Download Software window is displayed.

Step 2 In the Select a Product list, choose **Switches > Data Center Switches >** *platform* > *model* .

Step 3 If you are not already logged in as a registered Cisco user, you are prompted to log in now.

Step 4 From the Select a Software Type list, choose **NX-OS XML Schema Definition**.

Step 5 Find the desired release and click **Download.**

Step 6 If you are requested, follow the instructions to apply for eligibility to download strong encryption software images.

The Cisco End User License Agreement opens.

Step 7 Click **Agree** and follow the instructions to download the file to your PC.

NX-API Sandbox

The NX-API sandbox is a web-based user interface you use to enter the commands, command type, and output type for the Cisco Nexus 7000 Series device using HTTP or HTTPS. After posting the request, the output response is displayed.

By default, NX-API is disabled. Begin enabling NX-API with the **feature manager** command on the switch. Then enable NX-API with the **nxapi sandbox** command.

Use a browser to access the NX-API sandbox.

To view the command reference (that is, description of keywords) from the NX-API Web Interface, click the **Command Reference** link.

Note When using the NX-API sandbox, we recommend that you use the Firefox browser, Release 24.0 or later.

The following example shows how to configure and launch the NX-API sandbox:

- Enable the management interface:

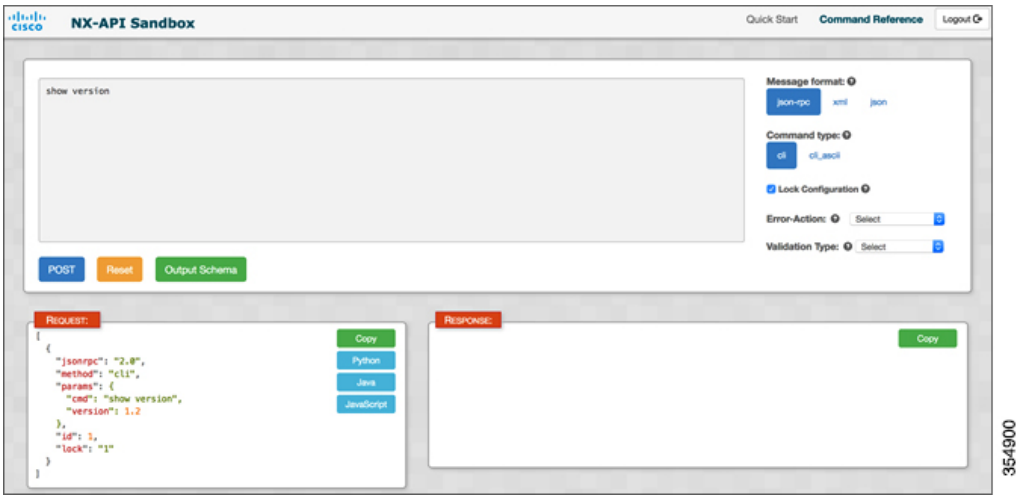
```
switch# configure terminal
switch(config)# interface mgmt 0
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

- Enable the nxapi feature:

```
switch# configure terminal
switch(config)# feature nxapi
switch(config)# nxapi sandbox
```

- Open a browser and enter *http://mgmt-ip* to launch the NX-API sandbox. The following figure is an example of a request and output response.

Figure 1: NX-API Sandbox with Example Request and Output Response



In the NX-API sandbox, specify the commands, command type, and output type in the top pane. Click **POST Request** button above the left pane to post the request. Brief descriptions of the request elements are displayed below the left pane.

After the request is posted, the output response is displayed in the right pane.

You can generate Java or JavaScript for each of the request posted through sandbox. To generate Java or Javascript code for each of the requests, click the Java or JavaScript button in the Request pane.

The following sections describe the commands that are available to manage NX-API, and descriptions of the elements in the request and the output responses.

NX-API Management Commands

You can enable and manage NX-API with the CLI commands listed in the following table.

Table 3: NX-API Management Commands

NX-API Management Command	Description
feature nxapi	Enables NX-API.
no feature nxapi	Disables NX-API.
nxapi {**http** | **https**} **port** *port*	Specifies a port.
no nxapi {**http** | **https**}	Disables HTTP or HTTPS.

NX-API Management Command	Description
show nxapi	Displays port information, NX-API enable or disable status, and sandbox enable or disable status.
nxapi sandbox	Enables NX-API sandbox.
no nxapi sandbox	Disables NX-API sandbox.
nxapi use-vrf *vrf-name*	Specifies the default VRF, management VRF, or named VRF.
nxapi certificate *certpath* **key** *keypath*	Specifies the upload of the following: • HTTPS certificate when *certpath* is specified. • HTTPS key when *keypath* is specified.

NX-API Request Elements

NX-API request elements are sent to the device in XML format, JSON format, or JSON-RPC format. The HTTP header of the request must identify the content type of the request.

Note A lock will be released by the system if the session that holds the lock is terminated. The session that acquired the lock can perform only necessary configurations.

When the input request format is XML or JSON, use the NX-API elements that are listed in the following table to specify a CLI command:

Table 4: NX-API Request Elements for XML or JSON Format

NX-API Request Element	Description
version	Specifies the NX-API version.

NX-API Request Element	Description
type	Specifies the type of command to be executed. The following command types are supported: • **cli_show** CLI **show** commands that expect structured output. If the command does not support XML output, an error message is returned. • **cli_show_ascii** CLI **show** commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes. • **cli_conf** CLI configuration commands. **Note** • Each command is executable only with the current user's authority. • The pipe operation is supported in the output when the message type is ASCII. If the output is in XML format, the pipe operation is not supported. • A maximum of 10 consecutive **show** commands are supported. If the number of **show** commands exceeds 10, the 11th and subsequent commands are ignored. • No interactive commands are supported.

NX-API Request Element	Description
chunk	Some **show** commands can return a large amount of output. For the NX-API client to start processing the output before the entire command is executed, NX-API supports output chunking for **show** commands. Enable or disable chunking with the following settings: **Note** • 0−Do not chunk output. • 1−Chunk output. **Note** • Only **show** commands support chunking. When a series of **show** commands are entered, only the first command is chunked and returned. • The output message format is XML. (This is the default.) Special characters, such as < or >, are converted to form a valid XML message (< is converted into < > is converted into &gt). • You can use XML SAX to parse the chunked output. • When chunking is enabled, the message format is limited to XML. The JSON output format is not supported when chunking is enabled.
roll_back	Specifies the configuration roll-back options. Specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in. **Note** The roll_back element is available in the cli_conf mode when the input request format is XML or JSON.

NX-API Request Element	Description
validate	Specifies the configuration validation settings. This element allows you to validate the commands before you apply them on the switch. This enables you to verify the consistency of a configuration, for example, the availability of necessary hardware resources, before applying it. Choose the validation type from the Validation Type drop-down list. • **Validate-Only**—Validates the configurations, but does not apply the configurations. • **Validate-and-Set**—Validates the configurations, and applies the configurations on the switch if validation is successful. **Note** The validate element is available in cli_conf mode when the input request format is XML or JSON.
lock	Allows you to specify an exclusive lock on the configuration whereby no other management or programming agent will be able to modify the configuration if this lock is held. **Note** The lock element is available in cli_conf mode when the input request format is XML or JSON.
sid	Specifies the session ID. This element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a *sid* to match the *sid* of the previous response message.
input	Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, **show** commands are cli_show message type and are not supported in cli_conf mode. **Note** Multiple commands are separated with a semi colon (;). The semi colon must be surrounded with single blank characters.) The following are examples of multiple commands: • cli_show–**show version**, **show interface brief**, **show vlan** • cli_conf–**interface Eth4/1**, **no shut**, **switchport**

NX-API Request Element	Description
output_format	The available output message formats are: **Note** • xml−Specifies output in XML format. • json−Specifies output in JSON format. **Note** The Cisco Nexus 7000 Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch. To manage the computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported. The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).

When JSON-RPC is the input request format, use the NX-API elements that are listed in the following table to specify a CLI command:

Table 5: NX-API Request Elements for JSON-RPC Format

NX-API Request Element	Description
jsonrpc	A string specifying the version of the JSON-RPC protocol. Version must be 2.0.
method	A string containing the name of the method to be invoked. NX-API supports either: • **cli**−show or configuration commands • **cli_ascii**−show or configuration commands; output without formatting
params	A structured value that holds the parameter values used during the invocation of a method. It must contain the following: • **cmd**−CLI command • **version**−NX-API request version identifier

NX-API Request Element	Description
roll_back	Configuration roll-back options. You can specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores the failed CLI and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
validate	Configuration validation settings. This element allows you to validate the commands before you apply them on the switch. This enables you to verify the consistency of a configuration (for example, the availability of necessary hardware resources) before applying it. Choose the validation type from the Validation Type drop-down list. • **Validate-Only**—Validates the configurations, but does not apply the configurations. • **Validate-and-Set**—Validates the configurations, and applies the configurations on the switch if the validation is successful.
lock	An exclusive lock on the configuration can be specified, whereby no other management or programming agent will be able to modify the configuration if this lock is held.
id	An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the *id* parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, *id* : 1

NX-API Response Elements

NX-API Response Elements for XML or JSON Requests

When the input request is in XML or JSON format, the response contains the following elements:

Table 6: NX-API Response Elements

NX-API Response Element	Description
version	NX-API version.

NX-API Response Element	Description
type	Type of command to be executed.
sid	Session ID of the response. This element is valid only when the response message is chunked.
outputs	Tag that encloses all command outputs. When multiple commands are in cli_show or cli_show_ascii, each command output is enclosed by a single output tag. When the message type is cli_conf, there is a single output tag for all the commands because cli_conf commands require context.
output	Tag that encloses the output of a single command output. For cli_conf message type, this element contains the outputs of all the commands.
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution. NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry at http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml.
msg	Error message associated with the returned error code.

NX-API Response Elements for JSON-RPC Requests

The response object of all JSON-RPC requests will be in JSON-RPC 2.0 response format as defined in the following table.

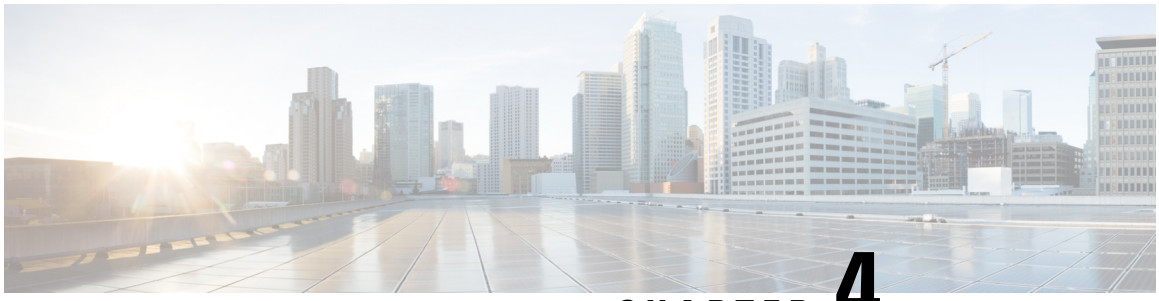
NX-API Response Element	Description
jsonrpc	A string specifying the version of the JSON-RPC protocol. The version is 2.0.
result	This field is included only upon receiving a successful response. The value of this field contains the requested CLI output.

NX-API Response Element	Description
error	This field is included only on error (mutually exclusive with the result object). The error object contains the following: • code — An integer error code as specified by the JSON-RPC specification. • message — A human-readable string to correspond with the error code. • data — An optional structure that contains useful information (such as CLI error messages or additional information).
id	The same value as the **id** in the corresponding request object. When there is a problem parsing the **id** in the request, this value is null.

Additional References

This section provides a list of sections that provide additional information about implementing NX-API:

- NX-API DevNet Community
- NX-API Github (Nexus 7000)
- NX-API Github (NX-OS Programmability scripts)



CHAPTER 4

Python API

Python API

Python is a programming language that has high-level data structures and a simple approach to object-oriented programming. Python's syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the standard library are freely available in source or binary form for all major platforms from the Python website:

http://www.python.org/

The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Cisco Nexus Series devices support Python version 2.7.5 in both interactive and noninteractive (script) modes is available in the Guest Shell.

The Python scripting capability provides programmatic access to the device's CLI to perform various tasks and PowerOn Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions. Python can also be accessed from the Bash shell.

The Python interpreter is available in the Cisco NX-OS software.

For information about using Python with Cisco Nexus devices, see the Cisco Nexus 9000 Series Python SDK User Guide and API Reference at this URL: http://opennxos.cisco.com/public/api/python/.

Using Python

This section describes how to write and execute Python scripts.

Python Package for Cisco

Cisco NX-OS provides a Python package for Cisco package that enables access to many core network device modules, such as interfaces, VLANs, VRFs, ACLs, and routes. You can display the details of the Python package for Cisco by entering the **help()** command. To obtain additional information about the classes and methods in a module, run the help command for a specific module, for example, **help** (*cisco.interface*) displays the properties of the cisco.interface module.

The following is an example of how to display information about the Python package for Cisco:

```
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
    cisco

FILE
    /isan/python/scripts/cisco/__init__.py

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

CLASSES
     __builtin__.object
        cisco.cisco_secret.CiscoSecret
        cisco.interface.Interface
        cisco.key.Key
```

Using the CLI Command APIs

The Python programming language uses three APIs that can execute CLI commands. These APIs are available from the Python CLI module.

These APIs are listed in the following table. You need to enable the APIs with the **from cli import** * command. The arguments for these APIs are strings of CLI commands. To execute a CLI command through the Python interpreter, enter the CLI command as an argument string of one of the following APIs:

Table 7: CLI Command APIs

API	Description
cli() Example: `string = cli ("cli-command")`	Returns the raw output of CLI commands, including control and special characters. **Note** The interactive Python interpreter prints control/special characters 'escaped'. A carriage return is printed as \n and gives results that might be difficult to read. The **clip()** API gives results that are more readable.

API	Description
clid() Example: `json_string = clid ("cli-command")`	Returns JSON output for **cli-command** if XML support exists for the command; otherwise, an exception is thrown. **Note** This API can be useful when searching the output of **show** commands.

clip()
Example:
`clip ("cli-command")`
	Prints the output of the **cli-command** directly to stdout and returns nothing to Python.
	Note `clip ("cli-command")`
	is equivalent to
	`r=cli("cli-command")`
	`print r`

To get output using Cisco/CLI library and read output from slot, enclose the complete CLI in double quotes and the argument in single quotes.

```
>>> cli("slot 6 quoted 'show hardware internal dev-port-map'")
```

When two or more commands are run individually, the state is not persistent from one command to subsequent commands.

In the following example, the second command fails because the state from the first command does not persist for the second command:

```
>>> cli("conf t")
>>> cli("interface eth4/1")
```

When two or more commands are run together, the state is persistent from one command to subsequent commands.

In the following example, the second command is successful because the state persists for the second and third commands:

```
>>> cli("conf t ; interface eth4/1 ; shut")
```

Note Commands are separated with a semicolon (;), as shown in the example. The semicolon must be surrounded with single blank characters.

Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python from the CLI. Note that the Python interpreter is designated with the >>> or … prompt.

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 5 ; no shut')
''
```

```
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...   intf=intflist['TABLE_interface']['ROW_interface'][i]
...   i=i+1
...   if intf['state'] == 'up':
...     print intf['interface']
...
mgmt0
Ethernet2/7
Ethernet4/7
loopback0
loopback5
>>>
```

Display Formats

The following examples show various display formats using Python APIs:

Example 1

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> clip('where detail')
mode:
username: admin
vdc: switch
routing-context vrf: default
```

Example 2

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
' mode: \n username: admin\n vdc:
switch\n routing-context vrf: default\n'
>>>
```

Example 3

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> r = cli('where detail') ; print r
mode:
username: admin
vdc: EOR-1
routing-context vrf: default
>>>
```

Example 4

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                kern_uptm_secs = 6
                kick_file_name = bootflash:///n9000-dk9.6.1.2.I1.1.bin
```

```
                    rr_service = None
                     module_id = Supervisor Module
                   kick_tmstmp = 10/21/2013 00:06:10
                bios_cmpl_time = 08/17/2013
                bootflash_size = 20971520
             kickstart_ver_str = 6.1(2)I1(2) [build 6.1(2)I1(2)] [gdb]
                kick_cmpl_time = 10/20/2013 4:00:00
                    chassis_id = Nexus9000 C9508 (8 Slot) Chassis
                 proc_board_id = SAL171211LX
                        memory = 16077872
                  manufacturer = Cisco Systems, Inc.
                kern_uptm_mins = 26
                  bios_ver_str = 06.14
                      cpu_name = Intel(R) Xeon(R) CPU E5-2403
                 kern_uptm_hrs = 2
                      rr_usecs = 816550
                   rr_sys_ver = None
                    rr_reason = Reset Requested by CLI command reload
                     rr_ctime = Mon Oct 21 00:10:24 2013
                   header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html
Copyright (c) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained herein are owned by other third parties and are
used and distributed under license.
Some parts of this software are covered under the GNU Public License. A copy of the license
 is available at
http://www.gnu.org/licenses/gpl.html.
                    host_name = switch
                     mem_type = kB
                kern_uptm_days = 0
>>>
```

Python script in Noninteractive Mode

A Python script can run in noninteractive mode when the Python script name is provided as an argument to the Python CLI command. Python scripts must be placed under the bootflash or volatile scheme. A maximum of 32 command-line arguments for the Python script are allowed with the Python CLI command.

Cisco Nexus 7000 Series devices also support the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

You can launch and run a python process in the backgorund. However, when the associated SSH terminal is terminated, the child processes started from the terminal will also be terminated. Beginning from release 8.0(1), if you want to run the script after terminal exit, ignore the SIGHUP signal in the running script. e.g. signal.signal(signal.SIGHUP, signal.SIG_IGN)

The following example shows a script and how to run it:

```
switch# show file bootflash:deltaCounters.py
#!/isan/bin/python

from cli import *
import sys, time

ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'

out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
```

```
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print 'row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast'
print '============================================================'
print '    %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc)
print '============================================================'

i = 0
while (i < count):
  time.sleep(delay)
  out = json.loads(clid(cmd))
  rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
  rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
  rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
  txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
  txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
  txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
  i += 1
  print '%-3d %8d %8d %8d %8d %8d %8d' % \
    (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew - rxbc, txucNew - txuc, txmcNew - txmc,
txbcNew - txbc)
```

```
switch# python bootflash:deltaCounters.py Ethernet1/1 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
============================================================
           0      791        1        0   212739        0
============================================================
1          0        0        0        0       26        0
2          0        0        0        0       27        0
3          0        1        0        0       54        0
4          0        1        0        0       55        0
5          0        1        0        0       81        0
switch#
```

The following example shows how a **source** command specifies command-line arguments. In this example, *policy-map* is an argument to the **cgrep python** script. The example also shows that a source command can follow after the pipe operator (|).

```
switch# show running-config | source sys/cgrep policy-map

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port
```

Running Scripts with the Embedded Event Manager

On Cisco Nexus 7000 Series devices, embedded event manager (EEM) policies support Python scripts.

The following example shows how to run a Python script as an EEM action:

- An EEM applet can include a Python script with an action command:

```
switch# show running-config eem

!Command: show running-config eem
```

```
!Time: Sun May  1 14:40:07 2011

version 6.1(2)I2(1)
event manager applet a1
  event cli match "show clock"
  action 1 cli python bootflash:pydate.py
  action 2 event-default
```

- You can search for the action triggered by an event in the log file by running the **show file** *logflash*:*event_archive_1* command:

```
switch# show file logflash:event_archive_1 | last 33

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
Python

2011-05-01 19:40:28.644891
Executing the following commands succeeded:
          python bootflash:pydate.py

PC_VSH_CMD_TLV(7679) with q
```

Python Integration with Cisco NX-OS Network Interfaces

On Cisco Nexus 7000 Series devices, Python is integrated with the underlying Cisco NX-OS network interfaces. You can switch from one virtual routing context to another by setting up a context through the cisco.vrf.set_global_vrf() API.

The following example shows how to retrieve an HTML document over the management interface of a device. You can also establish a connection to an external entity over the inband interface by switching to a desired virtual routing context.

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000

>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set_global_vrf in module cisco.vrf:

set_global_vrf(vrf)
    Sets the global vrf. Any new sockets that are created (using socket.socket)
    will automatically get set to this vrf (including sockets used by other
    python libraries).

    Arguments:
        vrf: VRF name (string) or the VRF ID (int).

    Returns: Nothing

>>>
```

Cisco NX-OS Security with Python

Cisco NX-OS resources are protected by the Cisco NX-OS sandbox layer of software and by CLI role-based access control (RBAC). Cisco NX-OS allows access to all resources, including file system, guest shell, and Bash commands for privileged users, which are limited to the network-admin and dev-ops roles.

For privileged users the sandbox is disabled. Python access for all other roles including custom are considered non-privileged and are contained by the sandbox. For more information on guidelines and limitations of VDC user roles, see Information About VDCs section in *Cisco Nexus 7000 Series Virtual Device Context Configuration Guide*.

- Examples of Security and User Authority, on page 30
- Example of Running Script with Scheduler, on page 31

Examples of Security and User Authority

The following example shows how a privileged user runs commands:

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
admin
0
>>> f=open('/tmp/test','w')
>>> f.write('hello from python')
>>> f.close()
>>> r=open('/tmp/test','r')
>>> print r.read()
hello from python
>>> r.close()
```

The following example shows a non-privileged user being denied access:

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')
Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

RBAC controls CLI access based on the login user privileges. A login user's identity is given to Python that is invoked from the CLI shell or from Bash. Python passes the login user's identity to any subprocess that is invoked from Python.

The following is an example for a privileged user:

```
>>> from cli import *
>>> cli('show clock')
'11:28:53.845 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf')
''
```

```
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Time: Sun May  8 11:29:40 2011

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

interface mgmt0
  vrf member management
vrf context blue
vrf context management
vrf context myvrf
```

The following is an example for a non-privileged user:

```
>>> from cli import *
>>> cli('show clock')
'11:18:47.482 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/isan/python/scripts/cli.py", line 20, in cli
    raise cmd_exec_error(msg)
errors.cmd_exec_error: '% Permission denied for the role\n\nCmd exec error.\n'
```

The following example shows an RBAC configuration:

```
switch# show user-account
user:admin
        this user account has no expiry date
        roles:network-admin
user:pyuser
        this user account has no expiry date
        roles:network-operator python-role
switch# show role name python-role
```

Example of Running Script with Scheduler

The following example shows a Python script that is running the script with the scheduler feature:

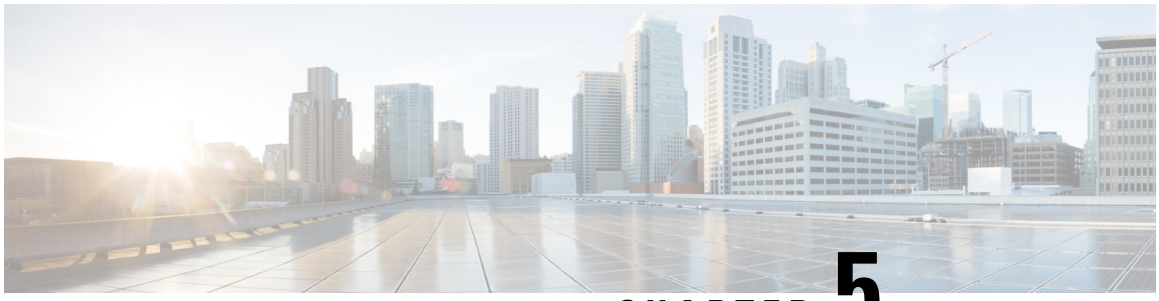
```
#!/bin/env python
from cli import *
from nxos import *
import os

switchname = cli("show switchname")
try:
    user = os.environ['USER']
except:
    user = "No user"
    pass

msg = user + " ran " + __file__ + " on : " + switchname
print msg
py_syslog(1, msg)
# Save this script in bootflash:///scripts
```

```
switch# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
```

```
switch(config)# feature scheduler
switch(config)# scheduler job name testplan
switch(config-job)# python bootflash:///scripts/testplan.py
switch(config-job)# exit
switch(config)# scheduler schedule name testplan
switch(config-schedule)# job name testplan
switch(config-schedule)# time start now repeat 0:0:4
Schedule starts from Mon Mar 14 16:40:03 2011
switch(config-schedule)# end
switch# term mon
2011 Mar 14 16:38:03 switch %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin on
10.19.68.246@pts/2
switch# show scheduler schedule
Schedule Name       : testplan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 4 Mins
Start Time          : Mon Mar 14 16:40:03 2011
Last Execution Time : Yet to be executed
----------------------------------------------
     Job Name            Last Execution Status
----------------------------------------------
    testplan                              -NA-
==============================================================================
switch#
switch# 2011 Mar 14 16:40:04 switch  %USER-1-SYSTEM_MSG: No user ran
/bootflash/scripts/testplan.py on : switch   - nxpython
2011 Mar 14 16:44:04 switch last message repeated 1 time
switch#
```



CHAPTER 5

XML Management Interface

XML Management Interface

This chapter describes how to use the XML management interface to configure devices.

Feature History for XML Management Interface

The following table lists the release history for this feature:

Table 8: Feature History for XML Management Interface

Feature Name	Releases	Feature Information
XML Management Interface	7.3(0)D1(1)	Added the following NetConf enhancements: • get-config • copy-config • validate • commit • lock • unlock • edit-config enhancements to support actions such as rollback on error, stop on error, continue on error, default operations, and candidate configuration.

Information About the XML Management Interface

You can use the XML management interface to configure a device. The interface uses the XML-based Network Configuration Protocol (NETCONF), which allows you to manage devices and communicate over the interface with an XML management tool or program. The Cisco NX-OS implementation of NETCONF requires you to use a Secure Shell (SSH) session for communication with a device.

NETCONF is implemented with an XML Schema (XSD) that allows you to enclose device configuration elements within a remote procedure call (RPC) message. From within an RPC message, select one of the NETCONF operations that matches the type of command that you want the device to execute. You can configure the entire set of CLI commands on the device with NETCONF. For information about using NETCONF, see the Creating NETCONF XML Instances, on page 38 and RFC 4741.

For more information about using NETCONF over SSH, see RFC 4742 .

This section includes the following topics:

NETCONF Layers

The following table lists the NETCONF layers:

Table 9: NETCONF Layers

Layer	Example
Transport protocol	SSHv2
RPC	rpc, rpc-reply
Operations	get-config, edit-config
Content	**show** or **configuration** command

The following is a description of the four NETCONF layers:

- SSH transport protocol—Provides an encrypted connection between a client and the server.
- RPC tag—Introduces a configuration command from the requestor and the corresponding reply from the XML server.
- NETCONF operation tag—Indicates the type of configuration command.
- Content—Indicates the XML representation of the feature that you want to configure.

SSH xmlagent

The device software provides an SSH service called xmlagent that supports NETCONF over SSH Version 2.

Note The xmlagent service is referred to as the XML server in Cisco NX-OS software.

NETCONF over SSH is initiated by the exchange of a Hello message between the client and the XML server. After the initial exchange, the client sends XML requests, which the server responds to with XML responses. The client and server terminate requests and responses with the character sequence >. Because this character sequence is not valid in XML, the client and the server can interpret when messages end, which keeps communication synchronized.

The XML schemas that define the XML configuration instances that you can use are described in Creating NETCONF XML Instances, on page 38.

Licensing Requirements for the XML Management Interface

Product	License Requirement
Cisco NX-OS	The XML management interface requires no license. Any feature that is not included in a license package is bundled with the Cisco NX-OS image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the *Cisco NX-OS Licensing Guide*.

Prerequisites to Using the XML Management Interface

Using the XML management interface has the following prerequisites:

- You must install SSHv2 on the client PC.
- You must install an XML management tool that supports NETCONF over SSH on the client PC.
- You must set the appropriate options for the XML server on the device.

Using the XML Management Interface

This section describes how to manually configure and use the XML management interface.

Note Use the XML management interface with the default settings on the device.

Configuring the SSH and the XML Server Options Through the CLI

By default, the SSH server is enabled on your device. If you disable SSH, you must enable it before you start an SSH session on the client PC.

You can configure the XML server options to control the number of concurrent sessions and the timeout for active sessions. You can also enable XML document validation and terminate XML sessions.

Note The XML server timeout applies only to active sessions.

For more information about configuring SSH, see http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/nx-os/security/configuration/guide/b_Cisco_Nexus_7000_NX-OS_Security_Configuration_Guide__Release_7-x.html.

For more information about the XML commands, see http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus7000/sw/system-management/guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide-RI.html.

Step 1 Enter global configuration mode.

configure terminal

Step 2 (Optional) Display information about XML server settings and active XML server sessions. You can find session numbers in the command output.

show xml server status

Step 3 Validate XML documents for the specified server session.

xml server validate all

Step 4 Terminate the specified XML server session.

xml server terminate *session*

Step 5 (Optional) Disable the SSH server so that you can generate keys.

no feature ssh

Step 6 Enable the SSH server. (The default is enabled.)

feature ssh

Step 7 (Optional) Display the status of the SSH server.

show ssh server

Step 8 Set the number of XML server sessions allowed.

xml server max-session *sessions*

The range is from 1 to 8. The default is 8.

Step 9 Set the number of seconds after which an XML server session is terminated.

xml server timeout *seconds*

The range is from 1 to 1200. The default is 1200 seconds.

Step 10 (Optional) Display information about the XML server settings and active XML server sessions.

show xml server status

Step 11 (Optional) Saves the running configuration to the startup configuration.

copy running-config startup-config

Example

The following example shows how to configure SSH and XML server options through the CLI:

```
switch# configure terminal
switch(config)# xml server validate all
switch(config)# xml server terminate 8665
switch(config)# no feature ssh
switch(config)# feature ssh server
```

```
switch(config)# xml server max-session 6
switch(config)# xml server timeout 2400
switch(config)# copy running-config startup-config
```

Starting an SSHv2 Session

You can start an SSHv2 session on a client PC with the **ssh2** command that is similar to the following:

```
ssh2 username@ip-address -s xmlagent
```

Enter the login username, the IP address of the device, and the service to connect to. The xmlagent service is referred to as the XML server in the device software.

Note The SSH command syntax might differ based on the SSH software on the client PC.

If you do not receive a Hello message from the XML server, verify the following conditions:

- The SSH server is enabled on the device.
- The XML server's *max-sessions* option is adequate to support the number of SSH connections to the device.
- The active XML server sessions on the device are not all in use.

Sending a Hello Message

You must advertise your capabilities to the server with a Hello message before the server processes any other requests. When you start an SSH session to the XML server, the server responds immediately with a Hello message that informs the client of the server's capabilities. The XML server supports only base capabilities and in turn expects support only for these base capabilities from the client.

The following are sample Hello messages from the server and the client:

Note You must end all XML documents with]]>]]> to support synchronization in NETCONF over SSH.

Hello Message from a Server

```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
  <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
  <session-id>25241</session-id>
</hello>]]>]]>
```

Hello Message from a Client

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

```
  </nc:capabilities>
</nc:hello>]]>]]>
```

Obtaining XML Schema Definition (XSD) Files

Step 1 From your browser, navigate to the Cisco software download site at:

http://software.cisco.com/download/navigator.html

The Download Software window is displayed.

Step 2 From the list of products displayed, choose **Switches > Data Center Switches >** *platform model*.

Step 3 If you are not already logged in as a registered Cisco user, you are prompted to log in now.

Step 4 From the **Select a Software Type** list, choose **NX-OS XML Schema Definition**.

Step 5 Find the desired release and click **Download.**

Step 6 If you are requested to, follow the instructions to apply for eligibility to download strong encryption software images.

The Cisco End User License Agreement is displayed.

Step 7 Click **Agree** and follow the instructions to download the file to your PC.

Sending an XML Document to the XML Server

To send an XML document to the XML server through an SSH session that you opened in a command shell, copy the XML text from an editor and paste it into the SSH session. Although typically you use an automated method to send XML documents to the XML server, you can verify the SSH connection to the XML server through this copy-paste method.

The following are the guidelines to follow when sending an XML document to the XML server:

- Verify that the XML server has sent the Hello message immediately after you started the SSH session, by looking for the Hello message text in the command shell output.
- Send the client Hello message before you send XML requests. Note that the XML server sends the Hello response immediately, and no additional response is sent after you send the client Hello message.
- Always terminate the XML document with the character sequence]]>]]>.

Creating NETCONF XML Instances

You can create NETCONF XML instances by enclosing the XML device elements within an RPC tag and NETCONF operation tags. The XML device elements are defined in feature-based XML schema definition (XSD) files, which enclose available CLI commands in an XML format.

The following are the tags used in the NETCONF XML request in a framework context. Tag lines are marked with the following letter codes:

- X —XML declaration
- R—RPC request tag
- N—NETCONF operation tags

- D—Device tags

NETCONF XML Framework Context

```
X <?xml version="1.0"?>
R <nc:rpc message-id="1" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
R xmlns="http://www.cisco.com/nxos:1.0:nfcli”>
N <nc:get>
N <nc:filter type="subtree">
D <show>
D <xml>
D <server>
D <status/>
D </server>
D </xml>
D </show>
N </nc:filter>
N </nc:get>
R </nc:rpc>]]>]]>
```

Note You must use your own XML editor or XML management interface tool to create XML instances.

RPC Request Tag

All NETCONF XML instances must begin with the RPC request tag <rpc>. The <rpc> element has a message ID (message-id) attribute. This message-id attribute is replicated in the <rpc-reply> and can be used to correlate requests and replies. The <rpc> node also contains the following XML namespace declarations:

- NETCONF namespace declaration—The <rpc> and NETCONF tags that are defined in the `urn:ietf:params:xml:ns:netconf:base:1.0` namespace, are present in the netconf.xsd schema file.
- Device namespace declaration—Device tags encapsulated by the <rpc> and NETCONF tags are defined in other namespaces. Device namespaces are feature oriented. Cisco NX-OS feature tags are defined in different namespaces. RPC Request Tag <rpc> is an example that uses the NFCLI feature. It declares that the device namespace is `xmlns=http://www.cisco.com/nxos:1.0:nfcli`. `nfcli.xsd` contains this namespace definition. For more information, see Obtaining XML Schema Definition (XSD) Files, on page 38.

Examples

RPC Request Tag <rpc>

```
<nc:rpc message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
...
</nc:rpc>]]>]]>
```

Configuration Request

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns="http://www.cisco.com/nxos:1.0:if_manager">
```

```
<nc:edit-config>
  <nc:target>
    <nc:running/>
  </nc:target>
  <nc:config>
    <configure>
      <__XML__MODE__exec_configure>
        <interface>
          <ethernet>
            <interface>2/30</interface>
            <__XML__MODE_if-ethernet>
              <__XML__MODE_if-eth-base>
                <description>
                  <desc_line>Marketing Network</desc_line>
                </description>
              </__XML__MODE_if-eth-base>
            </__XML__MODE_if-ethernet>
          </ethernet>
        </interface>
      </__XML__MODE__exec_configure>
    </configure>
  </nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>
```

Note __XML__MODE tags are used internally by the NETCONF agent. Some tags are present only as children of a certain __XML__MODE. By examining the schema file, you should be able to find the correct mode tag that leads to the tags representing the CLI command in XML.

NETCONF Operations Tags

NETCONF provides the following configuration operations:

Table 10: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 49
commit	Sets the running configuration to the current contents of candidate configuration.	NETCONF Commit Instance: Candidate Configuration Capability, on page 53
confirmed-commit	Provides the parameters to commit the configuration for a specified period of time. If this operation is not followed up by a commit operation within the confirm-timeout period, the configuration will be reverted to the state prior to the confirmed-commit operation.	NETCONF Confirmed Commit Instance, on page 54
copy-config	Copies the contents of the source configuration datastore to the target datastore.	NETCONF Copy Config Instance, on page 49

NETCONF Operation	Description	Example
delete-config	Operation not supported.	—
edit-config	Configures the features in the running configuration of the device. You use this operation for **configuration** commands. From Release 7.3(0)D1(1), support is added for the create, delete, and merge, rollback-on-error, continue-on-error, and stop-on-error actions.	NETCONF Edit Config Instance, on page 50 NETCONF Rollback-On-Error Instance, on page 54
get	Receives configuration information from a device. You use this operation for **show** commands. The source of the data is the running configuration.	Creating NETCONF XML Instances, on page 38
get-config	Retrieves all or part of a configuration.	NETCONF Get Config Instance, on page 52
kill-session	Closes the specified XML server session. Note that you cannot close your own session.	NETCONF Kill Session Instance, on page 49
lock	Allows a client to lock the configuration system of a device.	NETCONF Lock Instance, on page 52
unlock	Releases the configuration lock issued by the session.	NETCONF Unlock Instance, on page 53
validate	Checks a candidate's configuration for syntactical and semantic errors before applying the configuration to a device.	NETCONF Validate Capability Instance, on page 55

Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See Obtaining XML Schema Definition (XSD) Files, on page 38.

Using this schema, it is possible to build an XML instance. The relevant portions of the nfcli.xsd schema file that was used to build the NETCONF instances. See (Creating NETCONF XML Instances, on page 38).

show xml Device Tags

```
<xs:element name="show" type="show_type_Cmd_show_xml"/>
<xs:complexType name="show_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>to display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="xml" minOccurs="1" type="xml_type_Cmd_show_xml"/>
<xs:element name="debug" minOccurs="1" type="debug_type_Cmd_show_debug"/>
</xs:choice>
</xs:sequence>
<xs:attribute name="xpath-filter" type="xs:string"/>
```

```
<xs:attribute name="uses-namespace" type="nxos:bool_true"/>
</xs:complexType>
```

Server Status Device Tags

```
<xs:complexType name="xml_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent server</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

Device Tag Response

```
<xs:complexType name="status_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="__XML__OPT_Cmd_show_xml___readonly__" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:group ref="og_Cmd_show_xml___readonly__" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="og_Cmd_show_xml___readonly__">
<xs:sequence>
<xs:element name="__readonly__" minOccurs="1" type="__readonly___type_Cmd_show_xml"/>
</xs:sequence>
</xs:group>
<xs:complexType name="__readonly___type_Cmd_show_xml">
<xs:sequence>
<xs:group ref="bg_Cmd_show_xml_operational_status" maxOccurs="1"/>
<xs:group ref="bg_Cmd_show_xml_maximum_sessions_configured" maxOccurs="1"/>
<xs:group ref="og_Cmd_show_xml_TABLE_sessions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
```

Note The __XML__OPT_Cmd_show_xml___readonly__ tag is optional. This tag represents the response. For more information on responses, see RPC Response Tag, on page 47.

You can use the | XML option to find the tags that you should use to execute a <get> operation. The following is an example of the | XML option. This shows you that the namespace-defining tag used to execute operations

on this device is `http://www.cisco.com/nxos:1.0:nfcli` and that the nfcli.xsd file can be used to build requests.

You can enclose the NETCONF operation tags and the device tags within the RPC tag. The </rpc> end tag is followed by the XML termination character sequence.

XML Example

```
Switch#> show xml server status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:nfcli">
<nf:data>
<show>
<xml>
<server>
<status>
<__XML__OPT_Cmd_show_xml___readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>
<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML__OPT_Cmd_show_xml___readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Extended NETCONF Operations

Cisco NX-OS supports an <rpc> operation named <exec-command>. The operation allows client applications to send CLI **configuration** and **show** commands and to receive responses to those commands as XML tags.

The following is an example of the tags used to configure an interface. Tag lines are marked with the following letter codes:

- X —XML declaration
- R—RPC request tag
- EO—Extended operation

The following table provides a detailed explanation of the operation tags:

Table 11: Operation Tags

Tag	Description
<exec-command>	Executes a CLI command.

Tag	Description
<cmd>	Contains the CLI command. A command can be a **show** command or **configuration** command. Multiple configuration commands should be separated by a semicolon (;). Although multiple **show** commands are not supported. You can send multiple configuration commands in different <cmd> tags as part of the same request. For more information, see the Example on *Configuration CLI Commands Sent Through <exec-command>* .

Replies to configuration commands that are sent through the <cmd> tag are as follows:

- <nf:ok>—All **configuration** commands are executed successfully.
- <nf:rpc-error>—Some commands have failed. The operation stops at the first error, and the <nf:rpc-error> subtree provides more information about which configuration has failed. Note that configurations executed before the failed command would have been applied to the running configuration.

Configuration CLI Commands Sent Through the <exec-command>

The **show** command must be sent in its own <exec-command> instance as shown in the following example:

```
X <?xml version="1.0"?>
R <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
EO <nxos:exec-command>
EO <nxos:cmd>conf t ; interface ethernet 2/1 </nxos:cmd>
EO <nxos:cmd>channel-group 2000 ; no shut; </nxos:cmd>
EO </nxos:exec-command>
R </nf:rpc>]]>]]>
```

Response to CLI Commands Sent Through the <exec-command>

The following is the response to a send operation:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>
```

Show CLI Commands Sent Through the <exec-command>

The following example shows how the **show** CLI commands that are sent through the <exec-command> can be used to retrieve data:

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show interface brief</nxos:cmd>
```

```
</nxos:exec-command>
</nf:rpc>]]>]]>
```

Response to the show CLI Commands Sent Through the <exec-command>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0"
xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
<nf:data>
<mod:show>
<mod:interface>
<mod:__XML__OPT_Cmd_show_interface_brief___readonly__>
<mod:__readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
<mod:ip_addr>172.23.152.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML__OPT_Cmd_show_interface_brief___readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Failed Configuration

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 1.1.1.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
```

```
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

After a command is executed, the IP address of the interface is set, but the administrative state is not modified (the **no shut** command is not executed) because the **no port-channel 2000** command results in an error.

The <rpc-reply> as a result of a **show** command that is sent through the <cmd> tag contains the XML output of the **show** command.

You cannot combine configuration and show commands on the same <exec-command> instance. The following example shows **config** and **show** commands combined in the same instance.

Combination of configure and show Commands

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

show CLI Commands Sent Through the <exec-command>

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
```

```
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

NETCONF Replies

For every XML request sent by a client, the XML server sends an XML response enclosed in the RPC response tag <rpc-reply>.

RPC Response Tag

The following example shows the RPC response tag <rpc-reply>:

RPC Response Tag <rpc-reply>

```
<nc:rpc-reply message-id=”315” xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
<ok/>
</nc:rpc-reply>]]>]]>
```

RPC Response Elements

The elements <ok>, <data>, and <rpc-error> can appear in the RPC response. The following table describes the RPC response elements that can appear in the <rpc-reply> tag:

Table 12: RPC Response Elements

Element	Description
<ok>	The RPC request completed successfully. This element is used when no data is returned in the response.
<data>	The RPC request completed successfully. The data associated with the RPC request is enclosed in the <data> element.
<rpc-error>	The RPC request failed. Error information is enclosed in the <rpc-error> element.

Interpreting the Tags Encapsulated in the data Tag

The device tags encapsulated in the <data> tag contain the request, followed by the response. A client application can safely ignore all the tags before the <readonly> tag, as show in the following example:

RPC Reply Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>
```

```
<show>
<interface>
<__XML__OPT_Cmd_show_interface_brief___readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief___readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Note <__XML__OPT.*> and <__XML__BLK.*> appear in responses and are sometimes used in requests. These tags are used by the NETCONF agent and are present in responses after the <__readonly__> tag. They are necessary in requests, and should be added according to the schema file to reach the XML tag that represents the CLI command.

Example XML Instances

This section provides examples of the following XML instances:

NETCONF Close Session Instance

The following examples show the close-session request, followed by the close-session response:

Close Session Request

```
<?xml version="1.0"?>
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:close-session/>
</nc:rpc>]]>]]>
```

Close Session Response

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF Kill Session Instance

The following examples show the kill session request, followed by the kill session response:

Kill Session Request

```
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:kill-session>
<nc:session-id>25241</nc:session-id>
</nc:kill-session>
</nc:rpc>]]>]]>
```

Kill Session Response

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF Copy Config Instance

The following examples show the copy config request, followed by the copy config response:

Copy Config Request

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

```
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

Copy Config Response

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

Note <startup/> is not supported as a source or target datastore.

To perform any copy operation on startup-config like “copy running-config startup-config”, you need to fallback to the <exec-command> method.

NETCONF Edit Config Instance

The following examples show the use of NETCONF edit config:

Edit Config Request

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nc:edit-config>
<nc:target>
<nc:running/>
</nc:target>
<nc:config>
<configure>
<__XML__MODE__exec_configure>
<interface>
<ethernet>
<interface>2/30</interface>
<__XML__MODE_if-ethernet>
<__XML__MODE_if-eth-base>
<description>
<desc_line>Marketing Network</desc_line>
</description>
</__XML__MODE_if-eth-base>
</__XML__MODE_if-ethernet>
</ethernet>
</interface>
</__XML__MODE__exec_configure>
</configure>
</nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>
```

Edit Config Response

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager" message-id="16">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

The operation attribute in edit config identifies the point in configuration where the specified operation will be performed. If the operation attribute is not specified, the configuration is merged into the existing configuration data store. The operation attribute can have the following values:

- create
- merge
- delete

Edit Config: Delete Operation Request

The following example shows how to delete the configuration of interface Ethernet 0/0 from the running configuration:

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<default-operation>none</default-operation>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<top xmlns="http://example.com/schema/1.2/config">
<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

Response to Edit Config: Delete Operation

The following example shows how to edit the configuration of interface Ethernet 0/0 from the running configuration:

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

Note XML edit-config with candidate datastore is not supported with 1.0 version xml request. It is supported only with the newer version, which can be generated using xmlin tool.

NETCONF Get Config Instance

The following examples show the use of NETCONF get config:

Get Config Request to Retrieve the Entire Subtree

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<users/>
</top>
</filter>
</get-config>
</rpc>]]>]]>
```

Get Config Response with Results of a Query

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>
```

NETCONF Lock Instance

The following examples show a lock request, a success response, and a response to an unsuccessful attempt:

Lock Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<lock>
<target>
<running/>
</target>
</lock>
</rpc>]]>]]>
```

Response to a Successful Acquisition of Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/> <!-- lock succeeded -->
</rpc-reply>]]>]]>
```

Response to an Unsuccessful Attempt to Acquire Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error> <!-- lock failed -->
<error-type>protocol</error-type>
<error-tag>lock-denied</error-tag>
<error-severity>error</error-severity>
<error-message>
Lock failed, lock is already held
</error-message>
<error-info>
<session-id>454</session-id>
<!-- lock is held by NETCONF session 454 -->
</error-info>
</rpc-error>
</rpc-reply>]]>]]>
```

NETCONF Unlock Instance

The following examples show the use of NETCONF unlock:

Unlock Request

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

Response to an Unlock Request

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

NETCONF Commit Instance: Candidate Configuration Capability

The following examples show a commit operation and a commit reply:

Commit Operation

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

Commit Reply

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

NETCONF Confirmed Commit Instance

The following examples show a confirmed commit operation and a confirmed commit reply:

Confirmed Commit Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit>
<confirmed/>
<confirm-timeout>120</confirm-timeout>
</commit>
</rpc>]]>]]>
```

Confirmed Commit Response

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

NETCONF Rollback-On-Error Instance

The following examples show the how to configure rollback on error and the response to this request:

Rollback-On-Error Capability

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<error-option>rollback-on-error</error-option>
<config>
<top xmlns="http://example.com/schema/1.2/config">
<interface>
<name>Ethernet0/0</name>
<mtu>100000</mtu>
</interface>
```

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

Rollback-On-Error Response

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

NETCONF Validate Capability Instance

The following examples show the use of NETCONF validate capability that is identified by the string urn:ietf:params:netconf:capability:validate:1.0:

Validate Request

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>
```

Response to Validate Request

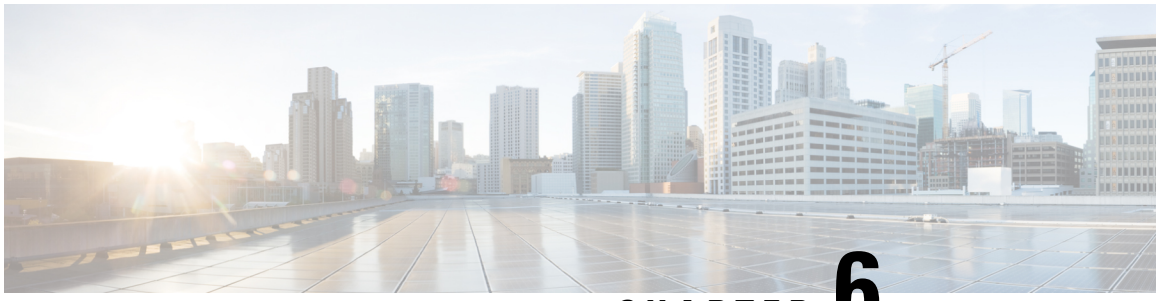
```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

Additional References

This section provides additional information related to implementing the XML management interface.

RFCs

RFCs	Title
RFC 4741	NETCONF Configuration Protocol
RFC 4742	Using the NETCONF Configuration Protocol over Secure Shell (SSH)



CHAPTER 6

Open Agent Container

Open Agent Container (OAC)

This chapter explains the Open Agent Container (OAC) environment and its installation in Cisco Nexus 7000 Series Switches. OAC is a 32-bit CentOS 6.7-based container that specifically allows open agents, such as the Puppet to run on these platforms.

Feature History for the Open Agent Container

This table lists the release history for this feature.

Table 13: Feature History for Open Agent Container

Feature Name	Releases	Feature Information
Open Agent Container (OAC)	8.4(1)	This feature is deprecated. Use of the virtual-service commands result in notifications about the deprecation.
Open Agent Container (OAC)	7.3(0)D1(1)	This feature was introduced in the Cisco Nexus 7000 Series Switches and Cisco Nexus 7700 Switches. The following commands were introduced or modified: **virtual-service**, **virtual-service connect**, **virtual-service install**, **virtual-service uninstall,** **virtual-service upgrade**, **show virtual-service list**, and **show virtual-service detail**.

Information About Open Agent Container

Beginning with Cisco NX-OS Release 8.4(1), the Open Agent Container support is deprecated. This featue was added in the Cisco NX-OS Release 7.3(0)D1(1) with the purpose of providing and execution space for configuration management. It is recommended to use agent-less configuration management systems such as Puppet or Ansible with the Cisco Nexus 7000 Series switches.

Open agents cannot be directly installed on these platforms. Instead, they run in a special environment—a decoupled execution space within a Linux Container (LXC)—called the Open Agent Container (OAC). Decoupling the execution space from the native host system allows customization of the Linux environment to suit the requirements of the applications without impacting the host system or applications running in other Linux containers.

The OAC is a 32-bit CentOS 6.7-based environment that provides a server-like experience to users. This means that after installation and first activation, users are responsible for setting up the DNS information in the /etc/resolv.conf or providing host information in the /etc/hosts, etc. as is done on any regular Linux system.

Note At a given point in time, OAC is supported in only one VDC.

By default, networking in the OAC is performed in the default routing table instance. Any additional route that is required (for example, a default route) must be configured in the native switch console and should not be configured using the CentOS commands. To use a different routing instance (for example, the management VRF), use the following commands:

To get a bash shell in the management VRF, run the **chvrf management** command.

To pass the VRF context to the specific command without changing the VRF instance in the shell, run the **chvrf management** *cmd* command.

Note The OAC occupies up to 256 MB of RAM and 400 MB of bootflash when enabled.

From within the OAC, the network administrator can perform the following functions:

- Access the network over Linux network interfaces.
- Access the device's volatile tmpfs.
- Access the device CLI using the **dohost** command.
- Access Cisco NX-API.
- Install and run Python scripts.
- Install and run 32-bit Linux applications.

Enabling OAC on your Switch

Installing and Activating Open Agent Container

The Open Agent Container (OAC) application software is packaged into a file with a .ova extension (OVA file, which will be hosted at the same location as the Cisco NX-OS images in the CCO directory and on GitHub). This package must first be copied to a location on the device using the **copy scp:**: command before it is installed on the device. The **install** keyword extracts the OVA file, validates the contents of the file, creates a virtual service instance, and validates the virtual machine definition file in XML. You do not have to copy configurations to the startup configuration file of the device to preserve the installation of the OVA file. After you download the oac.ova file to your device, install and activate the OAC. You can install a different OVA file on the active and standby route processors. To install and activate OAC on your device, perform the following.

Step 1 Add a virtual environment to the device:

switch# **virtual-service name** *virtual-service-name* **package** *package-location-media*

Note The media in which the package is located can be bootflash or any media, including a USB device.

Note Use the **show virtual-service list** command to view the progress of the installation. After the installation is complete, a message is displayed on the console informing you about the successful installation of the virtual service.

Step 2 After the installation is complete, enter global configuration mode and activate the virtual service:

switch# **configure terminal**

Step 3 Enable the NX-API feature:

switch(config)# **feature nxapi**

Communication between the Puppet agents and the Cisco Nexus devices is achieved using the NX-APIs.

Step 4 Configure the virtual service and enter virtual service configuration mode:

switch(config)# **virtual-service** *name*

Step 5 Activate the configured virtual service:

switch(config-virt-serv)# **activate**

Note To deactivate the virtual service, use the **no activate** command in virtual service configuration mode.

Step 6 Return to privileged EXEC mode:

switch(config-virt-serv)# **end**

Example

The following example shows how to install and activate the OAC in your Cisco NX-OS device. This is followed by the verification command that displays the details of the installed and configured virtual service.

```
switch# virtual-service install name oac package bootflash:oac.ova
switch# configure terminal
switch(config)# feature nxapi
switch(config)# virtual-service oac
switch(config-virt-serv)# activate
switch(config-virt-serv)# end
```

```
switch# show virtual-service detail

Virtual service oac detail
  State                 : Activated
  Package information
    Name                : oac.ova
    Path                : bootflash:/oac.ova
    Application
      Name              : OpenAgentContainer
      Installed version : 1.0
      Description       : Cisco Systems Open Agent Container
    Signing
      Key type          : Cisco release key
      Method            : SHA-1
    Licensing
      Name              : None
      Version           : None
  Resource reservation
    Disk                : 400 MB
    Memory              : 256 MB
    CPU                 : 1% system CPU

  Attached devices
    Type              Name        Alias
    ---------------------------------------------
    Disk              _rootfs
    Disk              /cisco/core
    Serial/shell
    Serial/aux
    Serial/Syslog                 serial2
    Serial/Trace                  serial3
```

Connecting to the Open Agent Container

To connect to the virtual service environment, use the **virtual-service connect name** *virtual-service-name* **console** command in privileged EXEC mode. In this case, the virtual environment we previously configured is the OAC.

```
switch# virtual-service connect name oac console
```

To access the OAC environment, use the following credentials:

username: **root**,

password: **oac**.

When you access the OAC environment for the first time, you will be prompted to reset your password immediately. Follow the instructions to reset your password. After you reset your password, you will have access to the OAC environment.

Note Press **Ctrl-C** thrice times to terminate the connection to the OAC and return to the switch console.

Verifying the Networking Environment Inside the Open Agent Container

To ensure that you can install open agents on your switch directly from the Internet, verify the networking environment within the configured OAC.

Step 1 Edit the /etc/resolv.conf to point to a DNS server.

The default servers are OpenDNS Public DNS (208.67.222.222 and 208.67.220.220).

Step 2 Make sure that you set the correct time in the container. You can set up the Network Time Protocol (NTP) on the host inside the VSH. The time from the host will automatically be synchronized with the OAC.

Step 3 If your switches are behind a firewall without direct connectivity to the internet use a proxy server.

Step 4 (Optional) Inside the container, set up http_proxy and https_proxy to point to your proxy server.

export http_proxy=<*your-http-proxy*>

export https_proxy=<*your-http-proxy*>

Upgrading Open Agent Container

If there is a new OVA available, you can upgrade the existing installation by using the **virtual-service upgrade name** *virtual-service-name* **package** *package-location-media* command in privileged EXEC mode. To upgrade to a new OVA, you must first deactivate the existing OVA by using the **no activate** command in virtual service configuration mode.

Caution After you upgrade, you will lose all the changes and configurations made in the earlier version of the OAC. You will have to start afresh in the new OAC environment.

Example

The following example shows you how to upgrade to a new OAC:

```
switch# configure terminal
switch(config)# feature nxapi
switch(config)# virtual-service oac
switch(config-virt-serv)# no activate
switch(config-virt-serv)# end
switch(config)# virtual-service install name oac package bootflash:oac1.ova
switch# configure terminal
switch(config)# feature nxapi
switch(config)# virtual-service oac
```

```
switch(config-virt-serv)# activate
switch(config-virt-serv)# end
```

Uninstalling Open Agent Container

To uninstall the OAC, perform the .

Before you begin

To uninstall the OAC from the Cisco NX-OS device, deactivate the OAC first.

Step 1 Enter global configuration mode and deactivate the virtual service:

switch# **configure terminal**

Step 2 Enter virtual service configuration mode:

switch(config)# **virtual-service** *virutal-service-name*

Step 3 Deactivate the configured virtual service:

switch(config-virt-serv)# **no activate**

Step 4 Exit to global configuration mode:

switch(config-virt-serv)# **exit**

Step 5 Disable the configured virtual service:

switch(config)# **no virtual-service** *virtual-service-name*

Step 6 Exit to privileged EXEC mode:

switch(config)# **exit**

Step 7 Uninstall the virtual service:

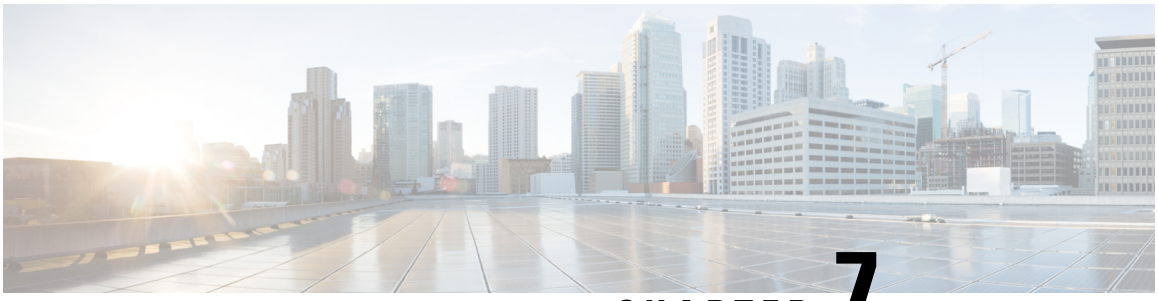
switch# **virtual-service uninstall name** *virtual-service-name*

Note Use the **show virtual-service list** command to view the progress of the uninstallation. After the uninstallation is complete, you will see a message on the console about the successful uninstallation of the virtual service.

Example:

The following example shows you how to deactivate and uninstall the OAC from your Cisco NX-OS device:

```
switch# configure terminal
switch(config)# virtual-service oac
switch(config-virt-serv)# no activate
switch(config-virt-serv)# exit
switch(config)# no virtual service oac
switch(config)# exit
switch# virtual-service uninstall name oac
```



CHAPTER 7

Using Puppet Agent with Cisco NX-OS

Puppet Agent with Cisco NX-OS

Feature History for Puppet Support

This table lists the release history for this feature:

Table 14: Feature History for Puppet Support

Feature Name	Releases	Feature Information
Puppet Agent with Cisco NX-OS	8.4(1)	This feature is deprecated. Use of the virtual-service commands result in notifications about the deprecation.
Puppet Agent with Cisco NX-OS	7.3(0)D1(1)	This feature was introduced in Cisco Nexus 7000 Series and Cisco Nexus 7700 switches.

Information About Puppet Agent

The Puppet software package, developed by Puppet Labs, is an open-source automation toolset for managing servers and other resources by enforcing device states, such as configuration settings.

Puppet components include a puppet agent that runs on the managed device (node) and a puppet master (server) that typically runs on a separate dedicated server and serves multiple devices. The operation of the puppet agent involves periodically connecting to the puppet master; which in turn compiles and sends a configuration manifest to the agent. The agent reconciles this manifest with the current state of the node and updates the state based on differences.

A puppet manifest is a collection of property definitions for setting the state on the device. The details for checking and setting these property states are abstracted so that a manifest can be used for more than one operating system or platform. Manifests are commonly used for defining configuration settings, but they can also be used to install software packages, copy files, and start services.

The following table lists resources that provide additional information about Puppet Labs:

Table 15: information on Puppet Labs

Puppet Labs	https://puppetlabs.com
Puppet Labs FAQ	http://docs.puppetlabs.com/guides/faq.html
Puppet Labs Documentation	http://docs.puppetlabs.com/

Prerequisites for Puppet Agent

The following are the prerequisites for the Puppet agent:

- You must have a Cisco device and an operating system software release that supports the installation.
  - Cisco Nexus 7000 Series Switch
  - Cisco Nexus 7700 Series Switch
  - Cisco NX-OS release 7.3(0)D1(1) or later for Cisco Nexus 7000 Series and Cisco Nexus 7700 series switches
- Puppet agents cannot run natively on Cisco Nexus 7000 Series and Cisco Nexus 7700 switches. Instead, they run in a special virtual environment called the OAC. For information on how to install OAC on your switch, refer to Open Agent Container, on page 57.
- You must have a Puppet primary server with Puppet 4.0 or later.
- You must have Puppet agent 4.0 or later.
- You must have ciscopuppet module 1.1.0 or later.

Puppet Agent in a Cisco NX-OS Environment

The Puppet agent software must be installed in a Linux environment on the Cisco Nexus platform.

Open Agent Container is a 32-bit CentOS 6.6-based container that is targeted to specifically allow Puppet agents on Cisco Nexus platforms. Although the container will have the ability to provide a Bash shell, it will restrict the applications that can be installed in the OAC.

Beginning with Cisco NX-OS Release 8.4(1), the puppet agent support is deprecated. This featue was added in the Cisco NX-OS Release 7.3(0)D1(1) with the purpose of providing and execution space for configuration management. It is recommended to use agent-less configuration management systems with the Cisco Nexus 7000 Series switches.

You have to download and install OAC on your device before you install the Puppet client on the device. For information about how to download and install OAC, refer to the chapter Open Agent Container (OAC), on page 57.

The following table provides information about agent software download, installation, and setup:

Table 16: Puppet Agent download, Installation and Setup Information

Puppet Agent RPM OAC, 32-bit CentOS6 root file)	http://yum.puppetlabs.com/ Release RPM is located in the repository under the name puppetlabs-release-el-6.noarch.rpm. For the latest information on Agent RPM, go to: https://github.com/cisco/cisco-network-puppet-module/tree/master#setup.
Puppet Agent: Installation & Setup on Cisco Nexus switches (Manual Setup)	Cisco Puppet Module::README-agent-install.md

ciscopuppet Module

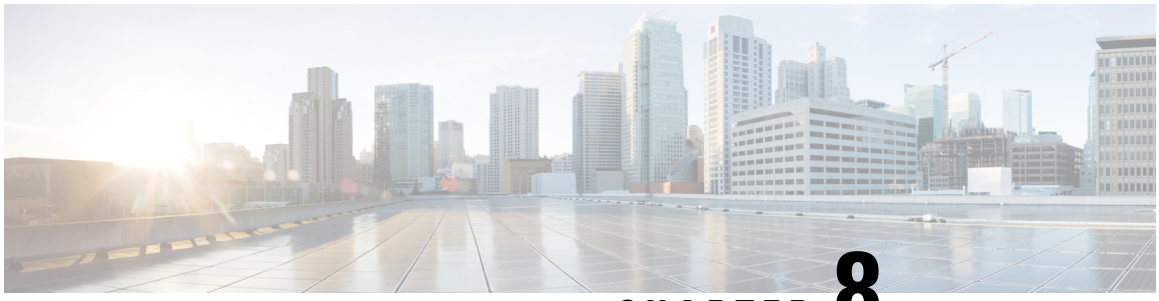
The ciscopuppet module is a Cisco-developed open-source interface between the abstract resources configuration in a puppet manifest and the specific implementation details of the Cisco NX-OS operating system and platform. This module is installed on the Puppet primary and is required for Puppet agent operations on Cisco Nexus switches.

The ciscopuppet module is available on Puppet Forge. For more information about the ciscopuppet module location and setup instructions, see:

https://forge.puppetlabs.com/puppetlabs/ciscopuppet

The following table contains links to documents that provide additional information about ciscopuppet module:

Topic	Link
Resource Type Catalog	https://github.com/cisco/cisco-network-puppet-module/tree/master#resource-by-tech
ciscopuppet Module: Source Code Repository	https://github.com/cisco/cisco-network-puppet-module/tree/master
ciscopuppet Module: Setup & Usage	Cisco Puppet Module::README.md
Puppet Labs: Installing Modules	https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html
Puppet Forge	https://forge.puppetlabs.com/



CHAPTER 8

Model-Driven Telemetry

This chapter contains the following sections:

About Telemetry

Collecting data for analyzing and troubleshooting has always been an important aspect in monitoring the health of a network.

Cisco NX-OS provides several mechanisms such as SNMP, CLI and Syslog to collect data from a network. These mechanisms have limitations that restrict automation and scale. One limitation is the use of the pull model, where the initial request for data from network elements originates from the client. The pull model does not scale when there is more than one network management station (NMS) in the network. With this model, the server sends data onlywhen clients request it. To initiate such requests, continual manual intervention is required. This continual manual intervention makes the pull model inefficient.

A push model continuously streams data out of the network and notifies the client. Telemetry enables the push model, which provides near-real-time access to monitoring data.

Telemetry Components and Process

Telemetry consists of four key elements:

- **Data Collection** — Telemetry data is collected from the database in branches of the object model specified using distinguished name (DN) paths. The data can be retrieved periodically (frequency- based). You can use the NX-API to collect frequency-based data.
- **Data Encoding** — The telemetry encoder encapsulates the collected data into the desired format for transporting.

  NX-OS encodes telemetry data in the Google Protocol Buffers (GPB).

- **Data Transport** — NX-OS transports telemetry data using the Google remote procedure call (gRPC) protocol for GPB encoding. The gRPC receiver supports message sizes greater than 4MB.
- **Telemetry Receiver** — A telemetry receiver is a remote management system or application that stores the telemetry data.

The GPB encoder stores data in a generic key-value format. The encoder requires metadata in the form of a compiled `.proto` file to translate the data into GPB format.

In order to correctly receive and decode the data stream, the receiver requires the `.proto` file that describes the encoding and the transport services. The encoding decodes the binary stream into a key value string pair.

A telemetry `.proto` file that describes the GPB encoding and gRPC transport is available on Cisco's GitLab: https://github.com/CiscoDevNet/nx-telemetry-proto

High Availability of the Telemetry Process

High availability of the telemetry process is supported with the following behaviors:

- **System Reload** — During system reload, any telemetry configuration and streaming services are restored.
- **Supervisor Failover** — Although telemetry is not on hot standby, telemetry configuration and streaming services are restored when the new active supervisor is running.
- **Process Restart** — If the telemetry process freezes or restarts for any reason, configuration and streaming services are restored when telemetry is restarted.

Licensing Requirements for Telemetry

Product	License Requirement
Cisco NX-OS	Telemetry requires no license. Any feature not included in a license package is bundled with the nx-os image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the *Cisco NX-OS Licensing Guide*.

Configuring Telemetry Using the NX-OS CLI

The following steps enables streaming telemetry, and configures the source and destination of the data stream.

SUMMARY STEPS

1. **openssl** *argument*
2. **configure terminal**
3. **feature telemetry**
4. **telemetry**
5. (Optional) **certificate** *certificate_path host_URL*
6. (Optional) **destination-profile**
7. **sensor-group** *sgrp_id*

8. (Optional) **data-source** *data-source-type*
9. **path** *sensor_path*
10. **destination-group** *dgrp_id*
11. (Optional) **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
12. (Optional) **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
13. *ip_version* **address** *ip_address* **port** *portnum*
14. (Optional) **use-chunking size** *chunking_size*
15. **subscription** *sub_id*
16. **snsr-grp** *sgrp_id* **sample-interval** *interval*
17. **dst-grp** *dgrp_id*

DETAILED STEPS

	Command or Action	Purpose

Step 1	**openssl** *argument*
	Example:
	Generate an SSL/TLS certificate using a specific argument, such as the following:
	• To generate a private RSA key: **openssl genrsa** *-cipher* **-out** *filename.key cipher-bit-length*
	For example:
	`switch# openssl genrsa -des3 server.key 2048`
	• To write the RSA key: **openssl rsa -in** *filename.key* **-out** *filename.key*
	For example:
	`switch# openssl rsa -in server.key -out server.key`
	• To create a certificate that contains the public/private key: **openssl req**
	-encoding-standard **-new** **-new** *filename.key* **-out** *filename.csr* **-subj '/CN=localhost'**
	For example:
	`switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'`
	• To create a public key: **openssl x509 -req** *-encoding-standard* **-days** *timeframe* **-in** *filename.csr* **-signkey** *filename.key* **-out** *filename.csr*
	For example:
		(Optional) Create an SSL/TLS certificate on the server that will receive the data, where `private.key` file is the private key and the `public.crt` is the public key.

	Command or Action	Purpose
	switch# **openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt**	
Step 2	**configure terminal** **Example:** switch# **configure terminal** switch(config)#	Enter the global configuration mode.
Step 3	**feature telemetry**	Enable the telemetry feature.
Step 4	**telemetry** **Example:** switch(config)# **telemetry** switch(config-telemetry)#	Enter configuration mode for telemetry.
Step 5	(Optional) **certificate** *certificate_path host_URL* **Example:** switch(config-telemetry)# **certificate /bootflash/server.key localhost**	Use an existing SSL/TLS certificate.

Step 6	(Optional) **destination-profile**
	Example:
	switch(config-telemetry)# **destination-profile**
	switch(conf-tm-dest-profile)# **use-vrf default**
	switch(conf-tm-dest-profile)# **use-compression gzip**
		• Specify a transport VRF and/or enable telemetry compression for gRPC transport.
		• Enter the **destination-profile** command to specify the default destination profile.
		• Enter any of the following commands:
		  • **use-vrf** *vrf* to specify the destination vrf.
		  • **use-compression gzip** to specify the destination compression method.
		After configuring the **use-vrf** command, you need to configure a new destination IP address within the new VRF. However, you may re-use the same destination IP address by un-configuring and re-configuring the destination. This ensures that the telemetry data streams to the same destination IP address in the new VRF.

Step 7	**sensor-group** *sgrp_id* **Example:** switch(config-telemetry)# **sensor-group 100** switch(conf-tm-sensor)#	Create a sensor group with ID *srgp_id* and enter sensor group configuration mode. Currently only numeric ID values are supported. The sensor group defines nodes that will be monitored for telemetry reporting.
Step 8	(Optional) **data-source** *data-source-type* **Example:** switch(config-telemetry)# **data-source NX-API**	Select a data source.

	Command or Action	Purpose
Step 9	**path** *sensor_path* **Example:** • The following command is applicable for DME, not for NX-API: `switch(conf-tm-sensor)# path <show command>`	Provide the show command to monitor as the path name.

Step 10	**destination-group** *dgrp_id*
	Example:
	`switch(conf-tm-sensor)# destination-group 100`
	`switch(conf-tm-dest)#`
		Create a destination group and enter destination group configuration mode.
		Currently *dgrp_id* only supports numeric ID values.

Step 11	(Optional) **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol gRPC encoding GPB`
		Specify an IPv4 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 12	(Optional) **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8002 protocol gRPC encoding GPB`
		Specify an IPv6 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 13	*ip_version* **address** *ip_address* **port** *portnum* **Example:** • For IPv4: `switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`	Create a destination profile for the outgoing data. When the destination group is linked to a subscription, telemetry data is sent to the IP address and port specified by this profile.
Step 14	(Optional) **use-chunking size** *chunking_size* **Example:** `switch(conf-tm-dest)# use-chunking size 64`	Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.

Step 15	**subscription** *sub_id*
	Example:
	`switch(conf-tm-dest)# subscription 100`
	`switch(conf-tm-sub)#`
		Create a subscription node with ID and enter the subscription configuration mode.
		Currently *sub_id* only supports numeric ID values.

	Command or Action	Purpose
Step 16	**snsr-grp** *sgrp_id* **sample-interval** *interval* **Example:** `switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000`	Link the sensor group with ID *sgrp_id* to this subscription and set the data sampling interval in milliseconds.
Step 17	**dst-grp** *dgrp_id* **Example:** `switch(conf-tm-sub)# dst-grp 100`	Link the destination group with ID *dgrp_id* to this subscription.

Configuration Examples for Telemetry Using the CLI

This example creates a subscription that streams data for the `sys/bgp` root MO every 5 seconds to the destination IP 1.2.3.4 port 50003.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path <show command>
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data every 5 seconds to destination IP 1.2.3.4 port 50003, and encrypts the stream using GPB encoding verified using the `test.pem`.

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path <show command>
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data every 15 seconds to destination IP 1.2.3.4 port 50004.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path <show command>
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a cadence-based collection of **show** command data every 750 seconds

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
```

```
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor# data-source NX-API
switch(conf-tm-sensor)# path "show system resources"
switch(conf-tm-sensor)# path "show version"
switch(conf-tm-sensor)# path "show environment power"
switch(conf-tm-sensor)# path "show environment fan"
switch(conf-tm-sensor)# path "show environment temperature"
switch(conf-tm-sensor)# path "show processes cpu"
switch(conf-tm-sensor)# path "show policy-map vlan"
switch(conf-tm-sensor)# path "show ip access-list test"
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

This example changes the sensor-group to frequency-based. After the following commands, the telemetry application will begin streaming the sys/fm data to the destination every 7 seconds.

```
switch(config)# telemetry
switch(config-telemetry)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
```

Multiple sensor groups and destinations can be linked to a single subscription. The subscription in this example streams the data for Ethernet port 1/1 to four different destinations every 10 seconds.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path <show command>
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol gRPC encoding GPB
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

A sensor group can contain multiple paths, a destination group can contain multiple destination profiles, and a subscription can be linked to multiple sensor groups and destination groups, as shown in this example.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path <path1>
switch(conf-tm-sensor)# path <path2>
switch(conf-tm-sensor)# path <path3>

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path <path4>
switch(conf-tm-sensor)# path <path5>

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path <path6>
switch(conf-tm-sensor)# path <path7>

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
```

```
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

switch(conf-tm-dest)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001

switch(conf-tm-dest)# destination-group 300
switch(conf-tm-dest)# ip address 1.2.3.4 port 60003

switch(conf-tm-dest)# subscription 600
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 200 sample-interval 20000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200

switch(conf-tm-dest)# subscription 900
switch(conf-tm-sub)# snsr-grp 200 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

You can verify the telemetry configuration using the **show running-config telemetry** command, as shown in this example.

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Time: Thu June 13 21:10:12 2018

version 8.3(1)
feature telemetry

telemetry
destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

Displaying Telemetry Configuration and Statistics

Use the following NX-OS CLI **show** commands to display telemetry configuration, statistics, errors, and session information.

show telemetry control database

This command displays the internal databases that reflect the configuration of telemetry.

```
switch# show telemetry control database ?
  <CR>
  >                   Redirect it to a file
  >>                  Redirect it to a file in append mode
  destination-groups  Show destination-groups
  destinations        Show destinations
```

```
  sensor-groups       Show sensor-groups
  sensor-paths        Show sensor-paths
  subscriptions       Show subscriptions
  |                   Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1

------------------------------------------------------------------------------
Subscription ID      Data Collector Type
------------------------------------------------------------------------------
100                  NX-API

Sensor Group Database size = 1

------------------------------------------------------------------------------
Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
------------------------------------------------------------------------------
100              Timer              10000(Running)         1

Sensor Path Database size = 1

------------------------------------------------------------------------------
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
------------------------------------------------------------------------------
No                       1              0           Full            <path>

Destination group Database size = 2

------------------------------------------------------------------------------
Destination Group ID  Refcount
------------------------------------------------------------------------------
100                   1

Destination Database size = 2

------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
------------------------------------------------------------------------------
192.168.20.111       12345      GPB        gRPC       1
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry control stats

This command displays the statistic regarding the internal databases regarding configuration of telemetry.

```
switch# show telemetry control stats
show telemetry control stats entered

------------------------------------------------------------------------------
Error Description                                              Error Count
------------------------------------------------------------------------------
Chunk allocation failures                                      0
Sensor path Database chunk creation failures                   0
Sensor Group Database chunk creation failures                  0
Destination Database chunk creation failures                   0
Destination Group Database chunk creation failures             0
Subscription Database chunk creation failures                  0
Sensor path Database creation failures                         0
Sensor Group Database creation failures                        0
Destination Database creation failures                         0
```

```
Destination Group Database creation failures                0
Subscription Database creation failures                     0
Sensor path Database insert failures                        0
Sensor Group Database insert failures                       0
Destination Database insert failures                        0
Destination Group Database insert failures                  0
Subscription insert to Subscription Database failures       0
Sensor path Database delete failures                        0
Sensor Group Database delete failures                       0
Destination Database delete failures                        0
Destination Group Database delete failures                  0
Delete Subscription from Subscription Database failures     0
Sensor path delete in use                                   0
Sensor Group delete in use                                  0
Destination delete in use                                   0
Destination Group delete in use                             0
Delete destination(in use) failure count                    0
Failed to get encode callback                               0
Sensor path Sensor Group list creation failures             0
Sensor path prop list creation failures                     0
Sensor path sec Sensor path list creation failures          0
Sensor path sec Sensor Group list creation failures         0
Sensor Group Sensor path list creation failures             0
Sensor Group Sensor subs list creation failures             0
Destination Group subs list creation failures               0
Destination Group Destinations list creation failures       0
Destination Destination Groups list creation failures       0
Subscription Sensor Group list creation failures            0
Subscription Destination Groups list creation failures      0
Sensor Group Sensor path list delete failures               0
Sensor Group Subscriptions list delete failures             0
Destination Group Subscriptions list delete failures        0
Destination Group Destinations list delete failures         0
Subscription Sensor Groups list delete failures             0
Subscription Destination Groups list delete failures        0
Destination Destination Groups list delete failures         0
Failed to delete Destination from Destination Group         0
Failed to delete Destination Group from Subscription        0
Failed to delete Sensor Group from Subscription             0
Failed to delete Sensor path from Sensor Group              0
Failed to get encode callback                               0
Failed to get transport callback                            0
switch#  Destination Database size = 1

------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
------------------------------------------------------------------------------
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry data collector brief

This command displays the brief statistic regarding the data collection.

```
switch# show telemetry data collector brief

----------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
----------------------------------------------------------------------
<xyz>                  143                        0
```

show telemetry data collector details

This command displays details statistic regarding the data collection which includes breakdown of all sensor paths.

```
switch# show telemetry data collector details

------------------------------------------------------------------------------
Succ Collections      Failed Collections     Sensor Path
------------------------------------------------------------------------------
150                   0                      <path>
```

show telemetry control pipeline stats

This command displays the statistic for the telemetry pipeline.

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
        Errors:
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0

Queue Statistics:
    Request Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

    Data Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
```

```
    Max Size          =     0    Full Count        =     0

Errors:
    Enqueue Error     =     0    Dequeue Error     =     0
```

show telemetry transport

This command displays all configured transport sessions.

```
switch# show telemetry transport

Session Id      IP Address      Port       Encoding   Transport  Status
--------------------------------------------------------------------------------
0               192.168.20.123   50001      GPB        gRPC       Connected
```

show telemetry transport <session-id>

This command displays detailed session information for a specific transport session.

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri May 02 11:45:57.505 UTC
Tx Error Count:      224
Last Tx Error:       Fri May 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

show telemetry transport <session-id> stats

This command displays details of a specific transport session.

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> errors

This command displays detailed error statistics for a specific transport session.

```
switch# show telemetry transport 0 errors

Session Id:                   0
Connection Stats
   Connection Count           1
   Last Connected:            Mon May 01 11:29:46.912 PST
   Disconnect Count           0
   Last Disconnected:         Never
Transmission Stats
   Transmit Count:            1225
   Last TX time:              Tue May 02 11:40:03.531 PST
   Min Tx Time:               7                 ms
   Max Tx Time:               1760              ms
   Avg Tx Time:               500               ms
```

Displaying Telemetry Log and Trace Information

Use the following NX-OS CLI commands to display the log and trace information.

show tech-support telemetry

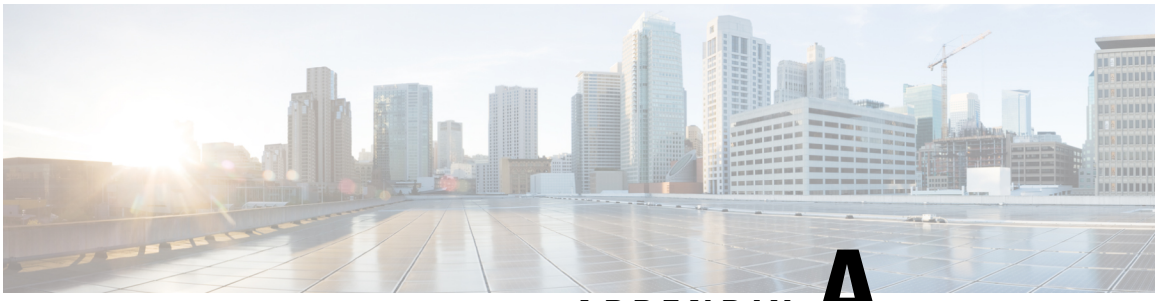
This NX-OS CLI command collects the telemetry log contents from the tech-support log. In this example, the command output is redirected into a file in bootflash.

```
switch# show tech-support telemetry > bootflash:tmst.log
```

Additional References

This section provides a list of sections that provide additional information about implementing NX-API:

- NX-API DevNet Community
- NX-API Github (Nexus 7000)
- NX-API Github (NX-OS Programmability scripts)



APPENDIX A

NX-API Response Codes

Table of NX-API Response Codes

When the request format is in XML or JSON format, the following are the possible NX-API errors, error codes, and messages of an NX-API response.

Note The standard HTTP error codes are at the Hypertext Transfer Protocol (HTTP) Status Code Registry: http://www.iana.org/assignments or http-status-codes http-status-codes.xhtml.

Table 17: NX-API Response Codes

NX-API Response	Code	Message
SUCCESS	200	Success.
CUST_OUTPUT_PIPED	204	Output is piped elsewhere due to request.
BASH_CMD_ERR	400	Input Bash command error.
CHUNK_ALLOW_ONE_CMD_ERR	400	Chunking allowed only to one command.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.
CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow **show**.
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.

EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive **show** commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Back-end processing error.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
BASH_CMD_NOT_SUPPORTED_ERR	501	**bash** command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

NX-API Response Codes for JSON-RPC Requests

When the request format is JSON-RPC, the following are the possible NX-API errors, error codes, and messages of an NX-API response:

Table 18: NX-API error for JSON-RPC request format

Code	Message	Meaning
-32700	Parse error.	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.

Code	Message	Meaning
-32600	Invalid request.	The JSON sent is not a valid request object.
-32601	Method not found.	The method does not exist or is not available.
-32602	Invalid parameters.	Invalid method parameters.
-32603	Internal error.	Internal JSON-RPC error.
-32000 to -32099	Server error.	Reserved for implementation-defined server errors.

NX-API Response Codes