Cisco Nexus 7000 Series NX-OS Programmability Guide

First Published: June 30, 2014

Last Modified: September 02, 2014

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

Text Part Number:

CONTENTS

Preface

Preface vii

Audience vii

Document Conventions vii

Related Documentation for Cisco Nexus 7000 Series NX-OS Software ix

Documentation Feedback xi

Obtaining Documentation and Submitting a Service Request xi

CHAPTER 1

New and Changed Information 1

New and Changed Information 1

CHAPTER 2

Overview 3

Programmability Overview 3

Standard Network Manageability Features 4

Advanced Automation Feature 4

PowerOn Auto Provisioning Support 4

OpenStack Integration 4

Programmability Support 6

NX-API Support 6

Python Scripting 6

Tcl Scripting 6

CHAPTER 3

NX-API 7

About NX-API 7

Transport 7

Message Format 7

Security 8

Using NX-API 8

Sending Requests 10
Obtaining the XSD Files 10
NX-API Sandbox 11
NX-API Management Commands 12
NX-API Request Elements 13
NX-API Response Elements 17
NX-API Response Elements for JSON-RPC Requests 18
Additional References 18

CHAPTER 4 Python API 19
About the Python API 19
Using Python 19
Cisco Python Package 19
Using the CLI Command APIs 20
Invoking the Python Interpreter from the CLI 21
Display Formats 22
Non-interactive Python 23
Running Scripts with Embedded Event Manager 24
Python Integration with Cisco NX-OS Network Interfaces 25
Cisco NX-OS Security with Python 25
Examples of Security and User Authority 25
Example of Running Script with Scheduler 26

CHAPTER 5 XML Management Interface 29
XML Management Interface 29
Feature History for XML Management Interface 29
About the XML Management Interface 30
NETCONF Layers 30
SSH xmlagent 30
Licensing Requirements for the XML Management Interface 31
Prerequisites to Using the XML Management Interface 31
Using the XML Management Interface 31
Configuring SSH and the XML Server Options Through the CLI 31
Starting an SSH Session 32
Sending the Hello Message 33

Obtaining the XSD Files 33
Sending an XML Document to the XML Server 34
Creating NETCONF XML Instances 34
RPC Request Tag rpc 35
NETCONF Operations Tags 36
Device Tags 37
Extended NETCONF Operations 40
NETCONF Replies 43
RPC Response Tag 44
Interpreting Tags Encapsulated in the Data Tag 44
Example XML Instances 45
NETCONF Close Session Instance 45
NETCONF Kill-session Instance 46
NETCONF copy-config Instance 46
NETCONF edit-config Instance 47
NETCONF get-config Instance 48
NETCONF Lock Instance 49
NETCONF unlock Instance 50
NETCONF Commit Instance - Candidate Configuration Capability 50
NETCONF Confirmed-commit Instance 51
NETCONF rollback-on-error Instance 51
NETCONF validate Capability Instance 52
Additional References 52

CHAPTER 6 **Open Agent Container** 55
Open Agent Container 55
Feature History for the Open Agent Container 55
About Open Agent Container 56
Enabling OAC on Your Switch 56
Installing and Activating the Open Agent Container 56
Connecting to the Open Agent Container 58
Verifying the Networking Environment Inside the Open Agent Container 58
Upgrading the OAC 59
Uninstalling the OAC 59

CHAPTER 7 Using Chef Client with Cisco NX-OS 61

Using Chef Client with Cisco NX-OS 61

Feature History for Chef Support 61

About Chef 61

Prerequisites 62

Chef Client NX-OS Environment 62

cisco-cookbook 63

CHAPTER 8 Using Puppet Agent with Cisco NX-OS 65

Using Puppet Agent with Cisco NX-OS 65

Feature History for Puppet Support 65

About Puppet 65

Prerequisites 66

Puppet Agent NX-OS Environment 66

ciscopuppet Module 67

APPENDIX A NX-API Response Codes 69

Table of NX-API Response Codes 69

NX-API Response Codes for JSON-RPC Requests 71

Preface

The Preface contains the following sections:

- Audience, page vii
- Document Conventions, page vii
- Related Documentation for Cisco Nexus 7000 Series NX-OS Software, page ix
- Documentation Feedback, page xi
- Obtaining Documentation and Submitting a Service Request, page xi

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).

Convention	Description
[x | y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{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:

http://www.cisco.com/en/us/products/ps9402/tsd_products_support_series_home.html

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:

http://www.cisco.com/en/us/products/ps9402/tsd_products_support_series_home.html

- *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 XML Interface User Guide*

Documentation Feedback

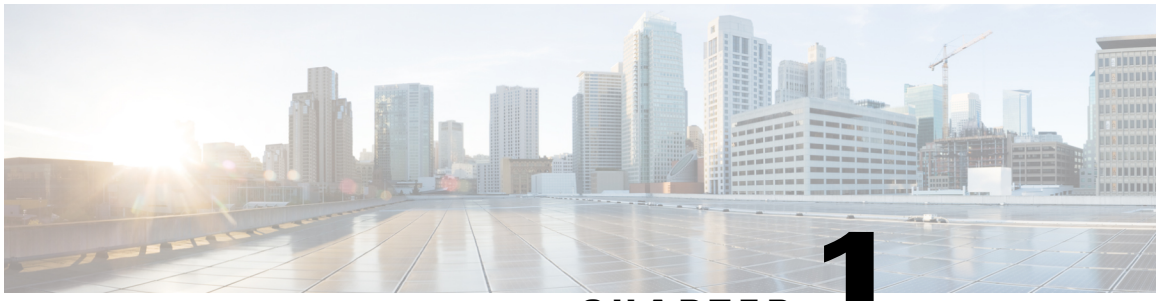
To provide technical feedback on this document, or to report an error or omission, please send your comments to: ciscodfa-docfeedback@cisco.com.

We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*, at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

To receive new and revised Cisco technical content directly to your desktop, you can subscribe to the What's New in Cisco Product Documentation RSS feed. RSS feeds are a free service.



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, page 1

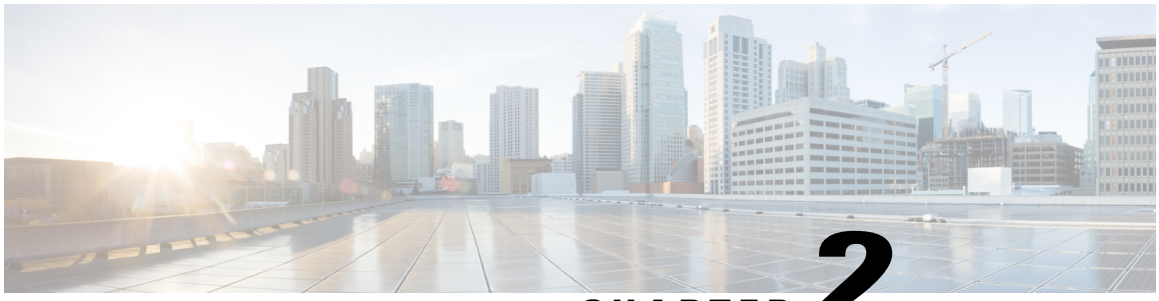
New and Changed Information

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

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
NX-API	This feature was introduced. NX-API improves the accessibility of CLIs that are run on the Cisco Nexus device, by making them available outside of the switch using HTTP/HTTPS.	7.2(0)D1(1)	NX-API, on page 7
Python API	This feature was introduced. The Python scripting capability gives programmatic access to the device's command-line interface (CLI) to perform various tasks and PowerOn Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions.	6.0(2)	Python API, on page 19
XML Management Interface	This feature was modified to support NETCONF Enhancements.	7.3(0)D1(1)	XML Management Interface, on page 29

Feature	Description	Changed in Release	Where Documented
Open Agent Container (OAC)	This feature was introduced. OAC is a 32-bit container that allows open agents to run on the Cisco Nexus 5600, 6000 and 7000 Series switches.	7.3(0)D1(1)	Open Agent Container, on page 55
Using Chef client with Cisco NX-OS	Support for Chef client on Cisco NX-OS introduced.	7.3(0)D1(1)	Using Chef Client with Cisco NX-OS, on page 61
Using Puppet agent with Cisco NX-OS	Support for Puppet agent introduced.	7.3(0)D1(1)	Using Puppet Agent with Cisco NX-OS, on page 65



CHAPTER 2

Overview

- Programmability Overview, page 3
- Standard Network Manageability Features, page 4
- Advanced Automation Feature, page 4
- Programmability Support, page 6

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).

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.
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.

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.

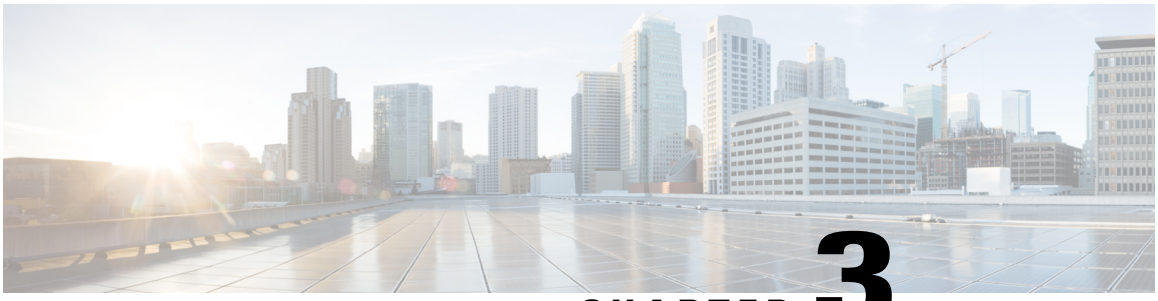
The Python scripting capability on the devices provide programmatic access to the switch CLI to perform various tasks, and to Power-On Auto Provisioning (POAP) and Embedded Event Manager (EEM) actions. Responses to Python calls that invoke the Cisco NX-OS CLI return text or JSON output.

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

- About NX-API, page 7
- Using NX-API, page 8
- Additional References, page 18

About NX-API

On Cisco Nexus devices, command-line interfaces (CLIs) are run only on the device. NX-API improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension 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.

Transport

NX-API uses HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/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. NX-API also supports JSON output format for specific commands.

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 HTTPS. All communication to the device is encrypted when you use HTTPS.

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

Note You should consider using HTTPS to secure your user's login credentials.

You can enable NX-API by using the **feature** manager CLI command. NX-API is disabled by default.

NX-API provides a session-based cookie, **nxapi_auth** when users first successfully authenticate. 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 performing the full 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 unnecessary use of the authentication process helps to reduce the workload on the device.

Note A **nxapi_auth** cookie expires in 600 seconds (10 minutes). This value is a fixed and cannot be adjusted.

Note 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

The commands, command type, and output type for the Cisco Nexus 7000 Series devices are entered using NX-API by encoding the CLIs into the body of a HTTP/HTTPs POST. The response to the request is returned in XML, JSON, or JSON-RPC output format.

Note For more details about NX-API response codes, see Table of NX-API Response Codes.

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

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

- Enable the management interface.

```
switch# conf t
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 NX-API **nxapi** feature.

```
switch# conf t
switch(config)# feature nxapi
```

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 NX-API requests via HTTP, use http://<ip-address-of-switch>/ins.

To send NX-API requests 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 equivalent to `application/json-rpc`. For the proprietary formats this should be either `text/xml` or `text/json` depending on the input format being used.

Obtaining the XSD Files

Step 1 From your browser, navigate to the Cisco software download site at the following URL: http://software.cisco.com/download/navigator.html

The Download Software page opens.

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

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

Use a browser to access the NX-API Sandbox.

Note When using the NX-API Sandbox, Cisco recommends 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.

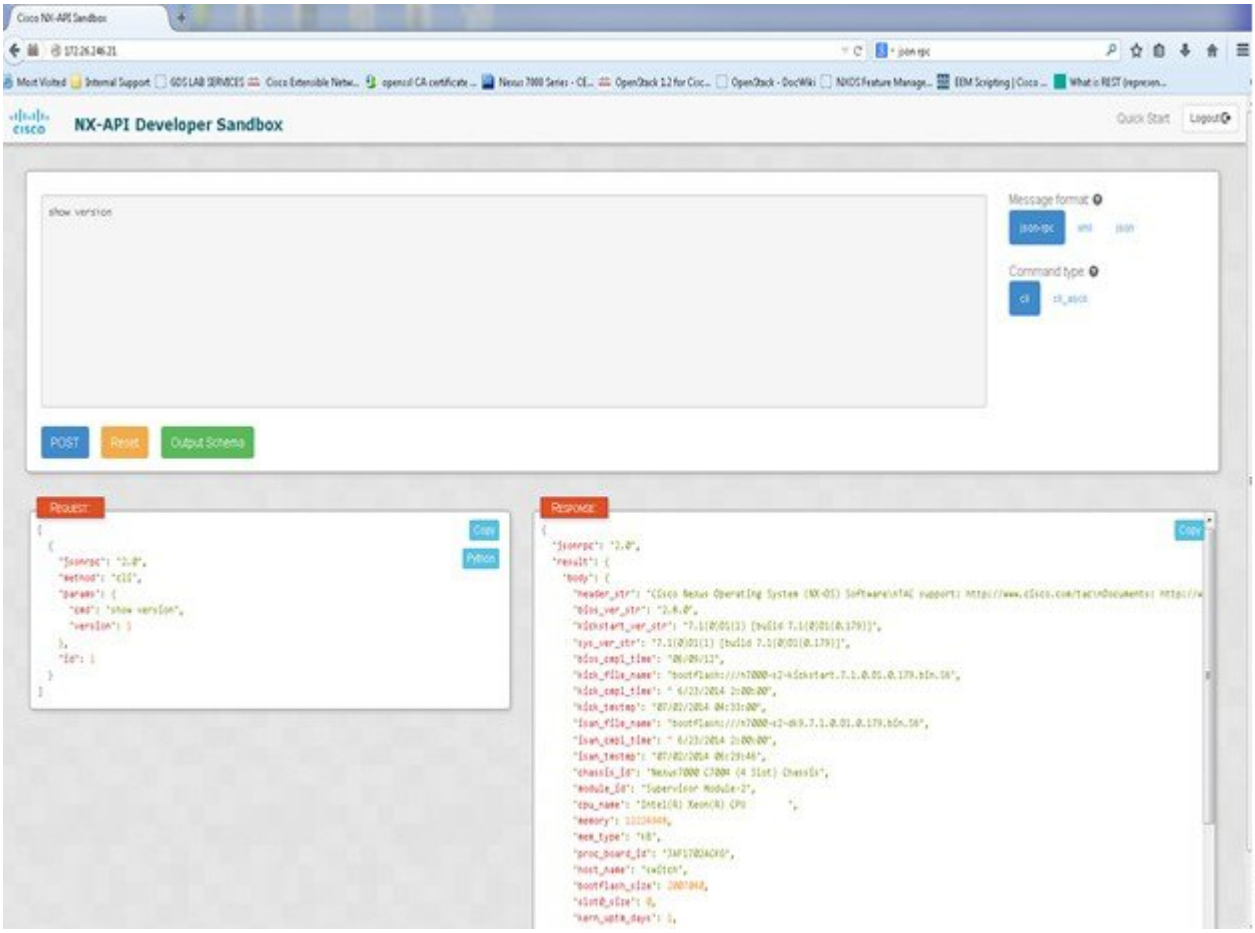
```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

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

```
switch# conf t
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, you specify the commands, command type, and output type in the top pane. Click the 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.

The following sections describe the commands to manage NX-API and descriptions of the elements of the request and the output response.

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.
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 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.

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

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 types of commands 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 only executable 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.

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>chunk</td><td>Some show commands can return a large amount of output. For the NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for show commands.
Enable or disable chunk with the following settings:<table><tr><td>0</td><td>Do not chunk output.</td></tr><tr><td>1</td><td>Chunk output.</td></tr></table>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. (XML is the default.) Special characters, such as < or >, are converted to form a valid XML message (< is converted into < > is converted into &gt).
Note You can use XML SAX to parse the chunked output. When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</td></tr>
<tr><td>sid</td><td>The session ID 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.</td></tr>
<tr><td>input</td><td>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 " ; ". (The ; must be surrounded with single blank characters.)
The following are examples of multiple commands:<table><tr><td>cli_show</td><td><code>show version ; show interface brief ; show vlan</code></td></tr><tr><td>cli_conf</td><td><code>interface Eth4/1 ; no shut ; switchport</code></td></tr><tr><td></td><td></td></tr></table></td></tr>
</table>

NX-API Request Element	Description
output_format	The available output message formats are the following: 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

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 the method. It must contain the following: • `cmd` a CLI command • `version` NX-API request version identifier

NX-API Request Element	Description
id	An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should normally not be null and numbers contain no fractional parts. If the user does not specify the *id* parameter, the server assumes the request is simply a notification resulting in a no response. For example, *id* : 1

NX-API Response Elements

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

Table 6: NX-API Response Elements

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

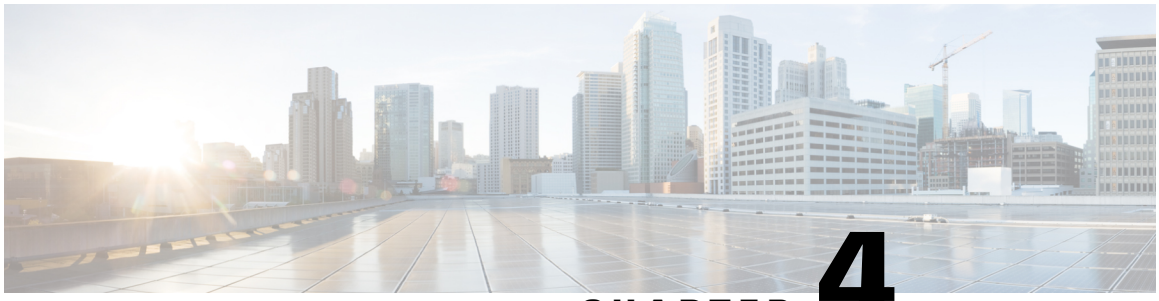
The response object of all JSON-RPC requests will be in JSON-RPC 2.0 response format as defined in the following table.

NX-API Response Element	Description
jsonrpc	A string specifying the version of the JSON-RPC protocol. Will be exactly `2.0`.
result	This field is only included on success. The value of this field contains the requested CLI output.
error	This field is only included 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 additional information related to implementing NX-API.

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



CHAPTER 4

Python API

- About the Python API , page 19
- Using Python, page 19

About the Python API

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant 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 extensive 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 7000 Series devices support Python v2.7.2 in both interactive and non-interactive (script) modes.

The Python scripting capability gives programmatic access to the device's command-line interface (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.

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

Cisco NX-OS provides a Cisco Python package. You can display the details of the Cisco Python package by entering the **help()** command. To obtain additional information about the classes and methods in a module,

you can 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 Cisco python package:

```
SWITCH# >>> import cisco
SWITCH# >>> help(cisco)
Help on module cisco:

NAME
    cisco - commands that integrate with CLI

FILE
    (built-in)

FUNCTIONS
    cli(...)
        execute a cli command

    clid(...)
        execute a cli command, return name/value pairs

    clip(...)
        execute a cli command, dont return it, just display it

    set_vrf(...)
        specify the vrf name for socket operations


SWITCH# >>>
SWITCH# >>> help(cisco.set_vrf)
Help on built-in function set_vrf in module cisco:

set_vrf(...)
    specify the vrf name for socket operations


SWITCH# >>>
```

Using the CLI Command APIs

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

These APIs are listed in the following table. The arguments for these APIs are strings of CLI commands. To execute a CLI command through the Python interpreter, you 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/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: `dictionary = clid("cli-command")`	**clid()**: d for 'dictionary' returns a dictionary of attribute-names/values for CLI commands that support XML. **Note** An exception is thrown when XML is not used. 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`

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 " ; " as shown in the example. (The ; 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 The Python interpreter is designated with the ">>>" or "…" prompt.

```
SWITCH# >>> cli('conf term ; interface loopback 1')
''
SWITCH(config-if)# >>> cli('ip address 1.1.1.1/24')
''
SWITCH(config-if)# >>> cli('exit')
''
SWITCH(config)# >>> cli('exit')
''
```

```
SWITCH# >>>
SWITCH# >>> a=clid('sh mod')
SWITCH# >>> a.keys()
['TABLE_modmacinfo/serialnum/4', 'TABLE_modinfo/modinf/1', 'TABLE_modmacinfo/serialnum/2',

'TABLE_modmacinfo/modmac/2', 'TABLE_modmacinfo/serialnum/1', 'TABLE_modinfo/ports/4',
'TABLE_modmacinfo/mac/2',
'TABLE_moddiaginfo/diagstatus/4', 'TABLE_modinfo/ports/1', 'TABLE_modinfo/ports/2',
'TABLE_modinfo/ports/3',
'TABLE_modwwninfo/modwwn/4', 'TABLE_xbarinfo/xbarports/1', 'TABLE_xbarinfo/xbarmodel/1',
'TABLE_modmacinfo/serialnum/3', 'TABLE_modwwninfo/hw/1', 'TABLE_modwwninfo/hw/3',
'TABLE_modwwninfo/hw/2',
'TABLE_moddiaginfo/diagstatus/3', 'TABLE_modwwninfo/hw/4', 'TABLE_modinfo/modinf/2',
'TABLE_modinfo/modinf/3',
'TABLE_modwwninfo/modwwn/1', 'TABLE_xbarwwninfo/xbarhw/1', 'TABLE_modinfo/modinf/4',
'TABLE_moddiaginfo/mod/3',
'TABLE_moddiaginfo/mod/2', 'TABLE_moddiaginfo/mod/1', 'TABLE_xbarinfo/xbartype/1',
'TABLE_xbarmacinfo/xbarmacaddr/1', 'TABLE_moddiaginfo/mod/4', 'TABLE_modinfo/status/4',
'TABLE_xbarwwninfo/xbarwwn/1', 'TABLE_modinfo/status/1', 'TABLE_modinfo/status/3',
'TABLE_modinfo/status/2',
'TABLE_modmacinfo/mac/1', 'TABLE_xbarwwninfo/xbarsw/1', 'TABLE_moddiaginfo/diagstatus/2',
'TABLE_modmacinfo/mac/3',
'TABLE_modinfo/modtype/2', 'TABLE_modinfo/modtype/3', 'TABLE_moddiaginfo/diagstatus/1',
'TABLE_modinfo/modtype/1',
'TABLE_modmacinfo/modmac/1', 'TABLE_modinfo/modtype/4', 'TABLE_modmacinfo/modmac/3',
'TABLE_xbarmacinfo/xbarserialnum/1', 'TABLE_xbarmacinfo/xbarmac/1',
'TABLE_xbarinfo/xbarinf/1',
'TABLE_modmacinfo/mac/4', 'TABLE_modwwninfo/sw/4', 'TABLE_modmacinfo/modmac/4',
'TABLE_modwwninfo/sw/1',
'TABLE_modwwninfo/sw/2', 'TABLE_modwwninfo/sw/3', 'TABLE_xbarinfo/xbarstatus/1',
'TABLE_modwwninfo/modwwn/2',
'TABLE_modinfo/model/4', 'TABLE_modinfo/model/3', 'TABLE_modinfo/model/2',
'TABLE_modinfo/model/1',
'TABLE_modwwninfo/modwwn/3']
SWITCH# >>> a["TABLE_modinfo/model/2"]
'N7K-SUP1'
SWITCH# >>>
```

Display Formats

The following examples show various display formats using the Python APIs:

Example 1:

```
SWITCH# >>> cli("conf ; interface loopback 1")
''
SWITCH(config-if)# >>> clip('where detail')
  mode:                conf
                       interface loopback1
  username:            admin
  vdc:                 SWITCH
  routing-context vrf: default
SWITCH(config-if)# >>>
```

Example 2:

```
SWITCH# >>> cli("conf ; interface loopback 1")
''
SWITCH(config-if)# >>>cli('where detail')
'  mode:                conf\n
                        interface loopback1\n
   username:            admin\n
   vdc:                 SWITCH\n
   routing-context vrf:  default\n'
SWITCH(config-if)# >>>
```

Example 3:

```
SWITCH# >>> cli("conf ; interface loopback 1")
''
```

```
SWITCH(config-if)# >>> r = cli('where detail') ; print r
  mode:                conf
                       interface loopback1
  username:            admin
  vdc:                 SWITCH
  routing-context vrf: default

SWITCH(config-if)# >>>
```

Example 4:

```
SWITCH# >>> r = clid('show version')
SWITCH# >>> for k in r.keys():
SWITCH# ...   print "%30s" % k, " = %s" % r[k]
SWITCH# ...
                kern_uptm_secs  = 58
                kick_file_name  = bootflash:///n7000-s1-kickstart.7.2.0.D1.1.gbin
                    rr_service  =
                     module_id  = Supervisor Module-1X
                    slot0_size  = 2044854
                   kick_tmstmp  = 06/14/2015 13:57:44
                isan_file_name  = bootflash:///n7000-s1-dk9.7.2.0.D1.1.gbin
                   sys_ver_str  = 7.2(0)D1(1) [gdb]
                bootflash_size  = 2000880
             kickstart_ver_str  = 7.2(0)D1(1) [gdb]
                kick_cmpl_time  =  5/19/2015 11:00:00
                    chassis_id  = Nexus7000 C7010 (10 Slot) Chassis
                 proc_board_id  = JAF1417DKEN
                        memory  = 4115196
                  manufacturer  = Cisco Systems, Inc.
                kern_uptm_mins  = 39
                  bios_ver_str  = 3.22.0
                      cpu_name  = Intel(R) Xeon(R) CPU
                bios_cmpl_time  = 02/20/10
                 kern_uptm_hrs  = 20
                      rr_usecs  = 228613
                  isan_tmstmp  = 06/14/2015 15:44:55
                   rr_sys_ver  = 7.3(0)ZD(0.114)
                    rr_reason  = Reset Requested by CLI command reload
                     rr_ctime  =  Fri Mar 15 20:41:38 2002

                   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) 2002-2015, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php

                isan_cmpl_time  =  5/19/2015 11:00:00
                     host_name  = SWITCH
                      mem_type  = kB
                kern_uptm_days  = 0
SWITCH# >>>
```

Non-interactive Python

A Python script can run in non-interactive mode by providing the Python script name as an argument to the Python CLI command. Python scripts must be placed under the bootflash/scripts directory for default vdc, and /bootflash/vdc_x/scripts for the non-default vdc (x).

The Cisco Nexus 7000 Series device also supports the source CLI command for running Python scripts. The **bootflash:scripts** directory is the default script directory for the source CLI command.

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

```
SWITCH# sh file bootflash:scripts/test1.py
#!/bin/env python
import os
import syslog
switchname = cli("show switchname")
try:
        user = os.environ['USER']
except:
        user = "No user"
        pass
msg = user + " ran " + __file__ + " on : " + switchname
print msg
syslog.syslog(1,msg)

SWITCH#
SWITCH# source test1.py
No user ran /bootflash/scripts/test1.py on : SWITCH

SWITCH# 2002 Mar 15 17:43:29
SWITCH %$ VDC-1 %$ %USER-1-SYSTEM_MSG: No user ran /bootflash/scripts/test1.py on : SWITCH
   - test1.py
```

The following example shows how a source command specifies command-line arguments. In the 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 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: Sat Mar 16 09:11:32 2002

version 7.2(0)D1(1)
event manager applet a2
  event cli match "show clock"
  action 1 cli command "source pydate.py"
  action 2 event-default
  action 3 syslog priority critical msg "$_cli_result"
```

- You can see the action triggered by the event by printing the **$_cli_result** command.

```
SWITCH# show clock
Time source is NTP
08:58:59.595 PST Sat Mar 16 2002
```

```
SWITCH#
2002 Mar 16 08:59:02 SWITCH %$ VDC-1 %$ eem_policy_dir:  %eem_policy_dir-2-LOG: a2:
Time source is NTP^M
2002 Mar 16 08:59:02 SWITCH %$ VDC-1 %$ eem_policy_dir: 08:59:02.269 PST Sat Mar 16
2002^M
2002 Mar 16 08:59:02 SWITCH %$ VDC-1 %$ eem_policy_dir: <Sat Mar 16 08:59:02 2002> <1>
 The System Manager library is unloading for PID 1985.^M
2002 Mar 16 08:59:02 SWITCH %$ VDC-1 %$ eem_policy_dir: ^M
```

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 set_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# >>> import urllib2
SWITCH# >>> set_vrf('management')
SWITCH# >>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
SWITCH# >>> print page.read()
Hello Cisco Nexus 7000

>>>


SWITCH# >>> import cisco
SWITCH# >>> help(cisco.set_vrf)
Help on built-in function set_vrf in module cisco:

set_vrf(...)
    specify the vrf name for socket operations


SWITCH# >>>
```

Cisco NX-OS Security with Python

Cisco NX-OS resources are protected by the Cisco NX-OS Sandbox layer of software and by the CLI role-based access control (RBAC).

All users associated with a Cisco NX-OS network-admin or dev-ops role are privileged users. Users who are granted access to Python with a custom role are regarded as non-privileged users. Non-privileged users have a limited access to Cisco NX-OS resources, such as file system, guest shell, and Bash commands. Privileged users have greater access to all the resources of Cisco NX-OS.

Examples of Security and User Authority

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

```
switch# python
switch# >>> from os import system
switch# >>> system("ls /isan/bin/")
-1
switch# >>> f=open("/bootflash/alias", "r")
switch# >>> f=open("/isan/bin/vsh", "r")
IOError: [Errno 1] Operation not permitted: '/isan/bin/vsh'
```

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:

```
SWITCH# python
Copyright (c) 2001-2012 Python Software Foundation; All Rights Reserved

SWITCH# >>> cli('show clock')
'Time source is NTP\n15:49:21.791 PST Fri Mar 15 2002\n'
SWITCH# >>>
```

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

```
User Access Verification
SWITCH login: name1
Password:
SWITCH# python
Copyright (c) 2001-2012 Python Software Foundation; All Rights Reserved

SWITCH# >>> cli('sh clock')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
cisco.cli_execution_error: % Permission denied for the role

SWITCH# >>>
```

The following example shows an RBAC configuration:

```
SWITCH(config-role)# show role name role-test

Role: role-test
  Description: new role
  Vlan policy: permit (default)
  Interface policy: permit (default)
  Vrf policy: permit (default)
  ----------------------------------------------------------------------
  Rule    Perm    Type        Scope               Entity
  ----------------------------------------------------------------------
  3       permit  command                         python
  2       deny    command                         sh running-config l3vm
  1       deny    command                         show clock
SWITCH(config-role)# exit
SWITCH(config)#
```

Example of Running Script with Scheduler

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

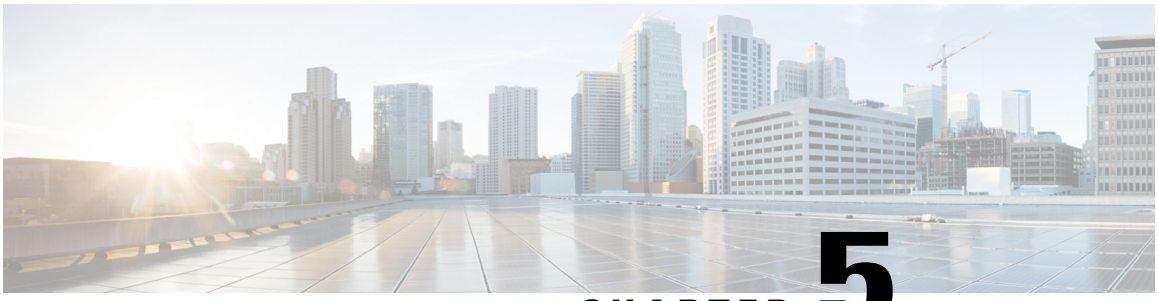
```
#!/bin/env python
import os
import syslog
switchname = cli("show switchname")
try:
 user = os.environ['USER']
except:
 user = "No user"
 pass
msg = user + " ran " + __file__ + " on : " + switchname
print msg
syslog.syslog(1,msg)

# Save this script in bootflash:///scripts
SWITCH# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SWITCH(config)# feature scheduler
SWITCH(config)# scheduler job name test-plan
SWITCH(config-job)# source test1.py
SWITCH(config-job)# exit
```

```
SWITCH(config)# scheduler schedule name test-plan
SWITCH(config-schedule)# job name test-plan
SWITCH(config-schedule)# time start now repeat 0:0:1
Schedule starts from Fri Mar 15 13:43:27 2002
SWITCH(config-schedule)# end
SWITCH# 2002 Mar 15 13:41:34
SWITCH %$ VDC-1 %$ %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin on console0

SWITCH# term mon
Console already monitors
SWITCH# show scheduler schedule
Schedule Name       : test-plan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 1 Mins
Start Time          : Fri Mar 15 13:43:27 2002
Last Execution Time : Yet to be executed
------------------------------------------------
     Job Name            Last Execution Status
------------------------------------------------
test-plan                                -NA-
================================================================================
SWITCH# 2002 Mar 15 13:43:27
SWITCH %$ VDC-1 %$ %USER-1-SYSTEM_MSG: No user ran /bootflash/scripts/test1.py on : SWITCH
   - test1.py

SWITCH# show scheduler schedule
Schedule Name       : test-plan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 1 Mins
Start Time          : Fri Mar 15 13:43:27 2002
Last Execution Time : Fri Mar 15 13:43:27 2002
Last Completion Time: Fri Mar 15 13:43:27 2002
Execution count     : 1
------------------------------------------------
     Job Name            Last Execution Status
------------------------------------------------
test-plan                           Success (0)
================================================================================
SWITCH#
```



CHAPTER 5

XML Management Interface

- XML Management Interface, page 29

XML Management Interface

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

Feature History for XML Management Interface

This table lists the release history for this feature.

Table 8: Feature History 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 like rollback on error, stop on error, continue on error, default operations and candidate config

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 the 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, you 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 34 and RFC 4741.

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

This section includes the following topics:

NETCONF Layers

The following are 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 a secure, 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 the 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 the messages end, which keeps communication synchronized.

The XML schemas that define XML configuration instances that you can use are described in the Creating NETCONF XML Instances, on page 34 section.

Licensing Requirements for the XML Management Interface

Product	License Requirement
Cisco NX-OS	The XML management interface requires no license. Any feature 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

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

Configuring SSH and the XML Server Options Through the CLI

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

You can configure 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 the Cisco NX-OS security configuration guide for your platform.

For more information about the XML commands, see the Cisco NX-OS system management configuration guide for your platform.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**configure terminal**	Enters global configuration mode.
Step 2	show **xml server status**	(Optional) Displays information about XML server settings and active XML server sessions. You can find session numbers in the command output.
Step 3	**xml server validate all**	Causes validation of XML documents for the specified server session.
Step 4	**xml server terminate** *session*	Terminates the specified XML server session.
Step 5	**no feature ssh**	(Optional) Disables the SSH server so that you can generate keys. For information about generating keys, see the Configuring SSH and the XML Server Options
Step 6	**feature ssh**	Enables the SSH server. The default is enabled.
Step 7	**show ssh server**	(Optional) Displays the status of the SSH server.
Step 8	**xml server max-session** *sessions*	Sets the number of allowed XML server sessions. The default is 8. The range is from 1 to 8.
Step 9	**xml server timeout** *seconds*	Sets the number of seconds after which the XML server session is terminated. The default is 1200 seconds. The range is from 1 to 1200.
Step 10	show **xml server status**	(Optional) Displays information about the XML server settings and active XML server sessions.
Step 11	copy running-config startup-config	(Optional) Saves the running configuration to the startup configuration.

The following example shows how to configure SSH and XML server options through 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 SSH Session

You can start an SSHv2 session on the client PC with a command 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 from 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 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 the Hello Message

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. You must advertise your capabilities to the server with a hello message before the server processes any other requests. The XML server supports only base capabilities and expects support only for the 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 the 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 the 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 the XSD Files

Step 1 From your browser, navigate to the Cisco software download site at the following URL: http://software.cisco.com/download/navigator.html

The Download Software page opens.

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.

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, you can 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 with this method.

Follow these guidelines for this method:

- Verify that the XML server 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 any XML requests. Because the XML server sends the hello response immediately, 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 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

Table 10: 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 rpc

All NETCONF XML instances must begin with the RPC request tag <rpc>. The example *RPC Request Tag <rpc>* shows the <rpc> element with its required **message-id** attribute. The 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 section on *Obtaining the XSD Files*.

Table 11: 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>]]>]]>
```

The following is an example of a configuration request.

Table 12: 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>]]>]]>
```

__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 13: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 45
commit	Sets the running configuration to current contents of candidate configuration.	NETCONF Commit Instance - Candidate Configuration Capability, on page 50
confirmed-commit	Provides parameters to commit the configuration for a specified period of time. If this operation is not followed by commit operation within confirm-timeout period, the configuration will be reverted to the state prior to the confirmed-commit operation.	NETCONF Confirmed-commit Instance , on page 51

NETCONF Operation	Description	Example
copy-config	Copies the content of source configuration datastore to the target datastore.	NETCONF copy-config Instance, on page 46
delete-config	Operation not supported.	—
edit-config	Configures features in the running configuration of the device. You use this operation for configuration commands. Starting Release 7.3(0)D1(1), support is added for actions - create, delete and merge; rollback-on-error, continue-on-error, stop-on-error.	NETCONF edit-config Instance, on page 47 NETCONF rollback-on-error Instance , on page 51
get	Receives configuration information from the device. You use this operation for **show** commands. The source of the data is the running configuration.	Creating NETCONF XML Instances, on page 34
get-config	Retrieves all or part of a configuration	NETCONF get-config Instance, on page 48
kill-session	Closes the specified XML server session. You cannot close your own session. See the close-session NETCONF operation.	NETCONF Kill-session Instance, on page 46
lock	Allows the client to lock the configuration system of a device.	NETCONF Lock Instance, on page 49
unlock	Releases the configuration lock issued by the session earlier.	NETCONF unlock Instance, on page 50
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	NETCONF validate Capability Instance , on page 52

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 the Obtaining the XSD Files, on page 33 section.

Using this schema, it is possible to build an XML instance. For example, the relevant portions of the nfcli.xsd schema file that was used to build Creating NETCONF XML Instances, on page 34 is shown in the following examples .

Table 14: 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>
```

Table 15: 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>
```

The following example shows the device tag response.

Table 16: 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 "__XML__OPT_Cmd_show_xml___readonly__" is optional. This tag represents the response. For more information on responses, see the RPC Response Tag, on page 44 section.

You can use the | XML option to find the tags you should use to execute a <get>. The following is an example of the | XML option.

Table 17: 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>
]]>]]>
```

From this response, you can see that the namespace defining tag to execute operations on this component is http://www.cisco.com/nxos:1.0:nfcli and the nfcli.xsd file can be used to build requests for this feature.

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.

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

Table 18: Configuration CLI Commands Sent Through <exec-command>

```
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>]]>]]>
```

The following is the response to the operation:

Table 19: Response to CLI Commands Sent Through <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" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>
```

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

Table 20: show CLI Commands Sent Through <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 interface brief</nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
```

The following is the response to the operation.

Table 21: Response to the show CLI commands Sent Through <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>
]]>]]>
```

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

Table 22: Tags

Tag	Description
<exec-command>	Executes a CLI command.
<cmd>	Contains the CLI command. A command can be a show or configuration command. Multiple configuration commands should be separated by a semicolon ";". 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 configure commands are executed successfully.
- <nf:rpc-error>: Some commands have failed. The operation stops on the first error, and the <nf:rpc-error> subtree provides more information on what configuration failed. Notice that any configuration executed before the failed command would have been applied to the running configuration.

The following example shows a failed configuration:

Table 23: 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>
]]>]]>
```

As a result of a command execution, 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 a configuration and **show** command that are combined in the same instance.

Table 24: Combination of Configuration 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>
]]>]]>
```

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

Table 25: Show CLI Commands Sent Through <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 the client, the XML server sends an XML response enclosed in the RPC response tag <rpc-reply>.

This section contains the following topics:

RPC Response Tag

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

Table 26: 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>]]>]]>
```

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 27: 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 Tags Encapsulated in the Data Tag

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

Table 28: 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>
]]>]]>
```

<__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 the examples of the following XML instances:

NETCONF Close Session Instance

The following example shows the close-session request, followed by the close-session response.

Table 29: 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>]]>]]>
```

Table 30: 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 example shows the kill-session request followed by the kill-session response.

Table 31: 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>]]>]]>
```

Table 32: 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 example shows the copy-config request followed by the copy-config response.

Table 33: 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>
```

Table 34: Copy-config Response

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

NETCONF edit-config Instance

The following example shows the use of NETCONF edit-config.

Table 35: 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>]]>]]>
```

Table 36: 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. Operation attribute can have the following values:

- create
- merge
- delete

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

Table 37: Edit-config: Delete Operation Request

```
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>]]>]]>
```

Table 38: Response to edit-config: Delete Operation

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

NETCONF get-config Instance

The following example shows the use of NETCONF get-config.

Table 39: 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>]]>]]>
```

Table 40: Get-config Response with Results of the 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 example shows the use of NETCONF lock operation.

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

Table 41: Lock Request

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

Table 42: Response to Successful Acquisition of Lock

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

Table 43: Response to Unsuccessful Attempt to Acquire the 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 example shows the use of NETCONF unlock operation.

Table 44: unlock request

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

Table 45: response to 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 example shows the commit operation and the commit reply:

Table 46: Commit Operation

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

Table 47: 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 example shows the confirmed-commit operation and confirmed-commit reply.

Table 48: 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>]]>]]>
```

Table 49: 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 example shows the use of NETCONF rollback on error capability that is identified by the string urn:ietf:params:netconf:capability:rollback-on-error:1.0

The following example shows the how to configure rollback on error and the response to this request.

Table 50: 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>]]>]]>
```

Table 51: 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 example shows the use of NETCONF validate capability that is identified by the string **urn:ietf:params:netconf:capability:validate:1.0**

Table 52: Validate request

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

Table 53: Response to validate request

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

Additional References

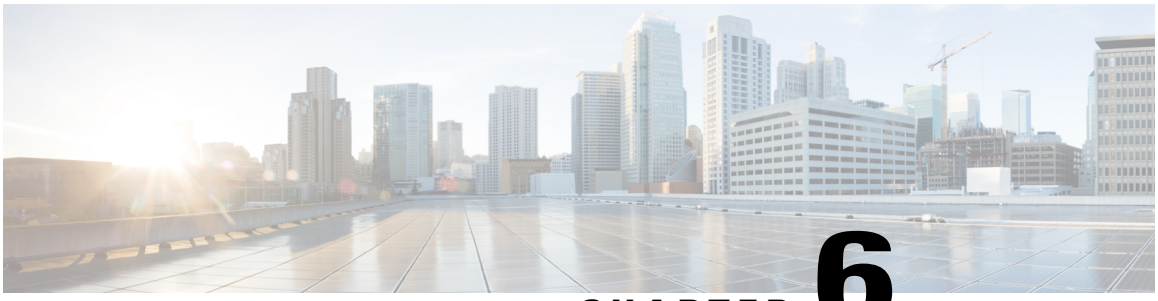
This section provides additional information related to implementing the XML management interface.

Standards

Standards	Title
No new or modified standards are supported by this feature. Support for existing standards has not been modified by this feature.	—

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, page 55

Open Agent Container

This chapter explains the Open Agent Container (OAC) environment and its installation in the Cisco Nexus 5600 Switches, Cisco Nexus 6000 Series Switches, and the Cisco Nexus 7000 Series Switches. OAC is a 32-bit CentOS 6.7-based container that specifically allows open agents, such as the Chef and Puppet agents to run on these platforms.

Feature History for the Open Agent Container

This table lists the release history for this feature.

Table 54: Feature History for Open Agent Container

Feature Name	Releases	Feature Information
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**.

About Open Agent Container

From Cisco NX-OS 7.3(0)D1(1) and later releases, the Cisco Nexus 7000 Series Switches and Cisco Nexus 7700 Switches support open agents, such as Chef and Puppet.

However, 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 /etc/hosts, etc. as is done on any regular Linux system.

By default, networking in the OAC is done 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 the 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 NXOS 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 don't have to copy configurations to the startup-configuration file of the device to preserve the installation of the OVA file. Once you download the oac.ova file on 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, do 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 other 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. Communication between the Puppet and Chef agents and the Nexus devices is achieved using the NX-APIs
switch(config)# **feature nxapi**

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, you must 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. Once you reset your password, you will have access to the OAC environment.

Note Press Ctrl-C three 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 /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 you will need to use a proxy server.

Step 4 Inside the container, setup the http_proxy and https_proxy to point to your proxy server. (This step is optional.)
export http_proxy="<*your-http-proxy*>"

export https_proxy="<*your-https-proxy*>"

Upgrading the OAC

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.

Note Once you upgrade, you will lose all changes and configurations made in old 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 the OAC

To uninstall the OAC from the NX-OS device, you must 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**

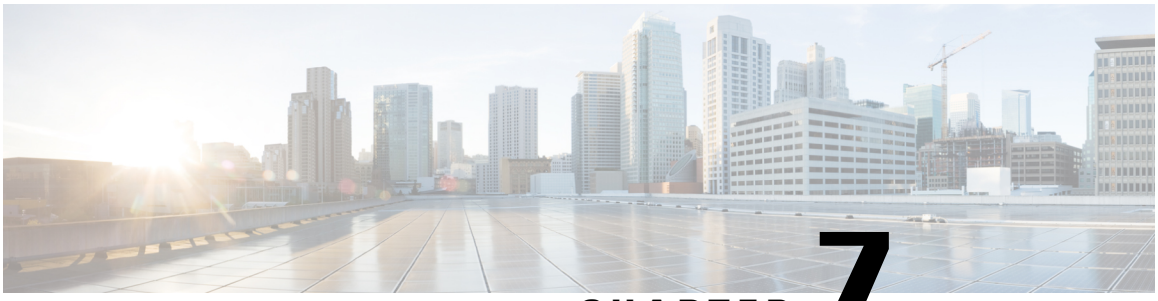
Step 7 Uninstall the virtual service.
switch# **virtual-service uninstall name** *virtual-service-name*

Note Use the **show virtual-service list** command to view the progress of the uninstallation. Once 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 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 Chef Client with Cisco NX-OS

- Using Chef Client with Cisco NX-OS, page 61

Using Chef Client with Cisco NX-OS

Feature History for Chef Support

This table lists the release history for this feature.

Table 55: Feature History for Chef Support

Feature Name	Releases	Feature Information
Support for Chef	7.3(0)D1(1)	This feature was introduced in Cisco Nexus 7000 and Cisco Nexus 7700 switches.

About Chef

Chef is an open-source software package developed by Chef Software, Inc. It is a systems and cloud infrastructure automation framework that deploys servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure. Each organization is comprised of one or more workstations, a single server, and every node that will be configured and maintained by the chef-client. Cookbooks and recipes are used to tell the chef-client how each node should be configured. The chef-client, which is installed on every node, does the actual configuration.

A Chef cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario, including libraries, recipes, files, and more. A Chef recipe is a collection of property definitions for setting state on the device. The details for checking and setting these property states are abstracted away so that a recipe may be used for more than one operating system or platform. While recipes are commonly used for defining configuration settings, they can also be used to install software packages, copy files, start services, and more.

The following references provide more information from Chef:

Topic	Link
Chef home	https://www.chef.io
Chef overview	https://docs.chef.io/chef_overview.html
Chef documentation (all)	https://docs.chef.io/

Prerequisites

The following are prerequisites for Chef:

- You must have a Cisco device and 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 and Cisco Nexus 7700 series switches.
- Chef agents cannot run natively on Cisco Nexus 7000 and Cisco Nexus 7700 switches. Instead they run in a special environment called the Open Agent Container (OAC). For information on how to install OAC on your switch, refer to the chapter Open Agent Container, on page 55
- You need a Chef server with Chef 12.4.1 or higher.
- You need Chef Client 12.4.1 or higher.
- You need Cisco Chef Cookbook Version 1.0.0. or later.

Chef Client NX-OS Environment

The chef-client software must be installed in a Linux environment provided by the Cisco Nexus platforms:

- Open Agent Container (OAC):

  It is a 32-bit CentOS 6.6 based container that is targeted to specifically allow Chef Agents on 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 Container.

You have to download and install OAC on your device before you install Chef on your device. For information on how to download and install OAC, refer to the chapter Open Agent Container, on page 55.

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

Topic	Link
Chef Client (OAC, 32-bit CentOS6 root file system)	For latest information on Client RPM, go to https://packagecloud.io/chef/stable/packages/el/6/chef-12.5.1-1.el6.i386.rpm.
Chef Client: Installation and setup on Cisco Nexus platform (manual setup)	cisco-cookbook::README-install-agent.md
Chef Client: Installation and setup on Cisco Nexus platform (automated installation using the Chef provisioner)	cisco-cookbook::README-chef-provisioning.md

cisco-cookbook

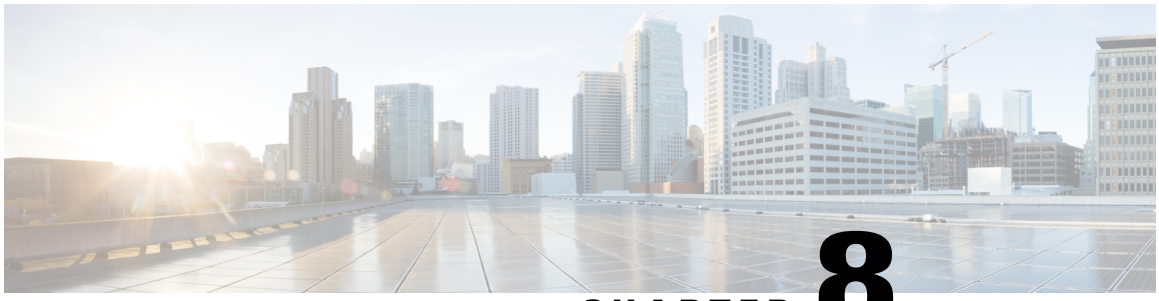
cisco-cookbook is a Cisco-developed open-source interface between the abstract resources configuration in a Chef recipe and the specific implementation details of the Cisco Nexus operating system and platforms. This cookbook is installed on the Chef Server and is required for proper Chef Client operation on Cisco Nexus devices.

cisco-cookbook can be found on Chef Supermarket. For information about cisco-cookbook location and setup instructions, see:

https://supermarket.chef.io/cookbooks/cisco-cookbook

The following table contains links to documents that provide additional information about cisco-cookbook:

Topic	Link
cisco-cookbook: Setup and usage	cisco-cookbook::README.md
cisco-cookbook: Source Code Repository	https://github.com/cisco/cisco-network-chef-cookbook/tree/master
Resource Type Catalog	https://github.com/cisco/cisco-network-chef-cookbook/tree/master#resource-reference
Chef Supermarket	https://supermarket.chef.io



CHAPTER 8

Using Puppet Agent with Cisco NX-OS

- Using Puppet Agent with Cisco NX-OS, page 65

Using Puppet Agent with Cisco NX-OS

Feature History for Puppet Support

This table lists the release history for this feature.

Table 56: Feature History for Puppet Support

Feature Name	Releases	Feature Information
Support for Puppet	7.3(0)D1(1)	This feature was introduced in Cisco Nexus 7000 and Cisco Nexus 7700 switches.

About Puppet

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 which 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 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.

More information can be found from 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

The following are prerequisites for the Puppet Agent:

- You must have a Cisco device and 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 and Cisco Nexus 7700 series switches.
- Puppet agents cannot run natively on Cisco Nexus 7000 and Cisco Nexus 7700 switches. Instead, they run in a special virtual environment called the Open Agent Container (OAC). For information on how to install OAC on your switch, refer to the chapter *Open Agent Container*.
- You must have Puppet Master 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 NX-OS Environment

The puppet agent software must be installed in a Linux environment on the Cisco Nexus platform:

- Open Agent Container (OAC):

  It is a 32-bit CentOS 6.6 based container that is targeted to specifically allow Puppet and Chef Agents on 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 Container.

You have to download and install OAC on your device before you install the Puppet client on the device. For information on how to download and install OAC, refer to the chapter Open Agent Container, on page 55.

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

Puppet Agent RPM (Open Agent Container (OAC), 32-bit CentOS6 root file)	http://yum.puppetlabs.com/ Release RPM is located in the repository with 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

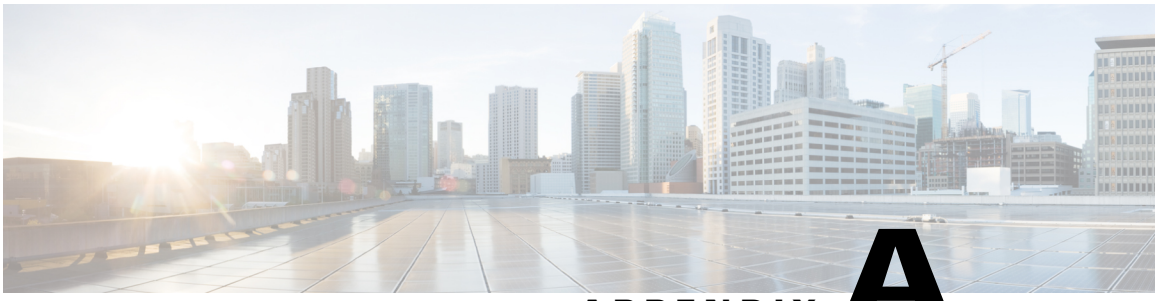
The **ciscopuppet** module is a Cisco developed open-source interface between the abstract resources configuration in a puppet manifest and the specific implementation details of the Cisco Nexus NX-OS operating system and platform. This module is installed on the puppet master and is required for puppet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

The **ciscopuppet** module is available on Puppet Forge. For more information about 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/



APPENDIX A

NX-API Response Codes

- Table of NX-API Response Codes, page 69

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/http-status-codes/http-status-codes.xhtml).

Table 57: 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 only allowed 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	Backend 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.

code	message	meaning
-32700	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-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.