



Cisco Nexus 5000 and 6000 Series NX-OS Programmability Guide

First Published: 2016-02-15

Last Modified: 2016-02-15

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:

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

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

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

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

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

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

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <http://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2016 Cisco Systems, Inc. All rights reserved.



CONTENTS

Preface

Preface vii

Audience vii

Document Conventions vii

Related Documentation for Cisco Nexus 5500 Series NX-OS Software ix

Related Documentation for Cisco Nexus 5600 Series NX-OS Software xi

Related Documentation for Cisco Nexus 6000 Series NX-OS Software xii

Documentation Feedback xiv

Obtaining Documentation and Submitting a Service Request xiv

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

Tel Scripting 6

CHAPTER 3

NX-API 9

About NX-API 9

Transport 9

Message Format 9

Security	10
Using NX-API	10
Sample NX-API Scripts	12
Obtaining the XSD Files	12
NX-API Sandbox	12
NX-API Management Commands	14
NX-API Request Elements	14
NX-API Response Elements	17
Command Types	18
Additional References	18

CHAPTER 4**Python API 21**

Information About the Python API	21
Installing Python	22
Installing Third Party Pure Python Packages	22
Using Python	22
Entering Python Shell	22
Executing Scripts	23
Passing Parameters to the Script	23
Application Programming Interface (API) Functions	23
Routes()	24
show_arp_table()	24
show_vsh_routes()	25
show_hw_routes()	26
verify_routes()	27
verify_arp_table()	27
CheckPortDiscards()	28
transfer()	29
CLI()	30
get_output()	31
rerun()	32
History()	32
get_history()	33
clear_history()	33

CHAPTER 5**XML Management Interface 35**

- XML Management Interface 35
- Feature History for XML Management Interface 35
- About the XML Management Interface 36
 - NETCONF Layers 36
 - SSH xmlagent 37
- Licensing Requirements for the XML Management Interface 37
- Prerequisites to Using the XML Management Interface 38
- Using the XML Management Interface 38
 - Configuring the SSH and the XML Server Options Through the CLI 38
 - Starting an SSHv2 Session 39
 - Sending a Hello Message 40
 - Obtaining XML Schema Definition (XSD) Files 40
 - Sending an XML Document to the XML Server 41
 - Creating NETCONF XML Instances 41
 - RPC Request Tag rpc 42
 - NETCONF Operations Tags 42
 - Device Tags 44
 - Extended NETCONF Operations 46
 - NETCONF Replies 49
 - RPC Response Tag 49
 - Interpreting the Tags Encapsulated in the Data Tag 50
- Example XML Instances 50
 - NETCONF Close Session Instance 51
 - NETCONF Kill Session Instance 51
 - NETCONF Copy Config Instance 52
 - NETCONF Edit Config Instance 52
 - NETCONF Get Config Instance 53
 - NETCONF Lock Instance 54
 - NETCONF Unlock Instance 55
 - NETCONF Commit Instance - Candidate Configuration Capability 55
 - NETCONF Confirmed Commit Instance 55
 - NETCONF Rollback-On-Error Instance 56
 - NETCONF Validate Capability Instance 56

Additional References 57

CHAPTER 6**Open Agent Container 59**

Open Agent Container 59

Feature History for the Open Agent Container 59

About Open Agent Container 60

Enabling OAC on Your Switch 60

Installing and Activating the Open Agent Container 60

Connecting to the Open Agent Container 62

Verifying the Networking Environment Inside the Open Agent Container 62

Upgrading the OAC 63

Uninstalling the OAC 63

CHAPTER 7**Using Chef Client with Cisco NX-OS 65**

Using Chef Client with Cisco NX-OS 65

Feature History for Chef Support 65

About Chef 65

Prerequisites 66

Chef Client NX-OS Environment 66

cisco-cookbook 67

CHAPTER 8**Using Puppet Agent with Cisco NX-OS 69**

Using Puppet Agent with Cisco NX-OS 69

Feature History for Puppet Support 69

About Puppet 69

Prerequisites 70

Puppet Agent NX-OS Environment 70

ciscopuppet Module 71

APPENDIX A**NX-API Response Codes 73**

Table of NX-API Response Codes 73

NX-API Response Codes for JSON-RPC Requests 75



Preface

The preface contains the following sections:

- [Audience, page vii](#)
- [Document Conventions, page vii](#)
- [Related Documentation for Cisco Nexus 5500 Series NX-OS Software, page ix](#)
- [Related Documentation for Cisco Nexus 5600 Series NX-OS Software, page xi](#)
- [Related Documentation for Cisco Nexus 6000 Series NX-OS Software, page xii](#)
- [Documentation Feedback, page xiv](#)
- [Obtaining Documentation and Submitting a Service Request, page xiv](#)

Audience

This publication is for network administrators who configure and maintain Cisco Nexus devices and Cisco Nexus 2000 Series Fabric Extenders.

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.

Convention	Description
<i>Italic</i>	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).
[x y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{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.
<i>italic screen font</i>	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 5500 Series NX-OS Software

The entire Cisco NX-OS 5500 Series documentation set is available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/tsd-products-support-series-home.html>

Release Notes

The release notes are available at the following URL:

http://www.cisco.com/en/US/products/ps9670/prod_release_notes_list.html

Configuration Guides

These guides are available at the following URL:

http://www.cisco.com/en/US/products/ps9670/products_installation_and_configuration_guides_list.html

The documents in this category include:

- *Cisco Nexus 5500 Series NX-OS Adapter-FEX Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS FabricPath Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS FCoE Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Fundamentals Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Interfaces Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Layer 2 Switching Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Multicast Routing Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Quality of Service Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS SAN Switching Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Security Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS System Management Configuration Guide*
- *Cisco Nexus 5500 Series NX-OS Unicast Routing Configuration Guide*

Installation and Upgrade Guides

These guides are available at the following URL:

http://www.cisco.com/en/US/products/ps9670/prod_installation_guides_list.html

The document in this category include:

- *Cisco Nexus 5500 Series NX-OS Software Upgrade and Downgrade Guides*

Licensing Guide

The *License and Copyright Information for Cisco NX-OS Software* is available at http://www.cisco.com/en/US/docs/switches/datacenter/sw/4_0/nx-os/license_agreement/nx-ossw_lisns.html.

Command References

These guides are available at the following URL:

http://www.cisco.com/en/US/products/ps9670/prod_command_reference_list.html

The documents in this category include:

- *Cisco Nexus 5500 Series NX-OS Fabric Extender Command Reference*
- *Cisco Nexus 5500 Series NX-OS FabricPath Command Reference*
- *Cisco Nexus 5500 Series NX-OS Fundamentals Command Reference*
- *Cisco Nexus 5500 Series NX-OS Interfaces Command Reference*
- *Cisco Nexus 5500 Series NX-OS Layer 2 Interfaces Command Reference*
- *Cisco Nexus 5500 Series NX-OS Multicast Routing Command Reference*
- *Cisco Nexus 5500 Series NX-OS Quality of Service Command Reference*
- *Cisco Nexus 5500 Series NX-OS Security Command Reference*
- *Cisco Nexus 5500 Series NX-OS System Management Command Reference*
- *Cisco Nexus 5500 Series NX-OS TrustSec Command Reference*
- *Cisco Nexus 5500 Series NX-OS Unicast Routing Command Reference*
- *Cisco Nexus 5500 Series NX-OS Virtual Port Channel Command Reference*

Technical References

The *Cisco Nexus 5500 Series NX-OS MIB Reference* is available at http://www.cisco.com/en/US/docs/switches/datacenter/nexus5500/sw/mib/reference/NX5500_MIBRef.html.

Error and System Messages

The *Cisco Nexus 5500 Series NX-OS System Message Guide* is available at http://www.cisco.com/en/US/docs/switches/datacenter/nexus5500/sw/system_messages/reference/sl_nxos_book.html.

Troubleshooting Guide

The *Cisco Nexus 5500 Series NX-OS Troubleshooting Guide* is available at http://www.cisco.com/en/US/docs/switches/datacenter/nexus5500/sw/troubleshooting/guide/N5K_Troubleshooting_Guide.html.

Related Documentation for Cisco Nexus 5600 Series NX-OS Software

The entire Cisco NX-OS 5600 Series documentation set is available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/tsd-products-support-series-home.html>

Release Notes

The release notes are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/products-release-notes-list.html>

Configuration Guides

These guides are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/products-installation-and-configuration-guides-list.html>

The documents in this category include:

- *Cisco Nexus 5600 Series NX-OS Adapter-FEX Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS FabricPath Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Fibre Channel over Ethernet Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Fundamentals Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Interfaces Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Layer 2 Switching Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Multicast Routing Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Quality of Service Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS SAN Switching Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Security Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS System Management Configuration Guide*
- *Cisco Nexus 5600 Series NX-OS Unicast Routing Configuration Guide*

Licensing Guide

The *License and Copyright Information for Cisco NX-OS Software* is available at http://www.cisco.com/en/US/docs/switches/datacenter/sw/4_0/nx-os/license_agreement/nx-oss_w_lisns.html.

Command References

These guides are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/products-command-reference-list.html>

The documents in this category include:

- *Cisco Nexus 5600 Series NX-OS Fabric Extender Command Reference*
- *Cisco Nexus 5600 Series NX-OS FabricPath Command Reference*
- *Cisco Nexus 5600 Series NX-OS Fibre Channel Command Reference*
- *Cisco Nexus 5600 Series NX-OS Fundamentals Command Reference*
- *Cisco Nexus 5600 Series NX-OS Interfaces Command Reference*
- *Cisco Nexus 5600 Series NX-OS Layer 2 Interfaces Command Reference*
- *Cisco Nexus 5600 Series NX-OS Multicast Routing Command Reference*
- *Cisco Nexus 5600 Series NX-OS QoS Command Reference*
- *Cisco Nexus 5600 Series NX-OS Security Command Reference*
- *Cisco Nexus 5600 Series NX-OS System Management Command Reference*
- *Cisco Nexus 5600 Series NX-OS TrustSec Command Reference*
- *Cisco Nexus 5600 Series NX-OS Unicast Routing Command Reference*
- *Cisco Nexus 5600 Series NX-OS Virtual Port Channel Command Reference*

Error and System Messages

The *Cisco Nexus 5600 Series NX-OS System Message Guide* is available at http://www.cisco.com/en/US/docs/switches/datacenter/nexus5500/sw/system_messages/reference/sl_nxos_book.html.

Troubleshooting Guide

The *Cisco Nexus 5600 Series NX-OS Troubleshooting Guide* is available at <http://www.cisco.com/c/en/us/support/switches/nexus-5000-series-switches/products-troubleshooting-guides-list.html>.

Related Documentation for Cisco Nexus 6000 Series NX-OS Software

The entire Cisco NX-OS 6000 Series documentation set is available at the following URL:

http://www.cisco.com/en/US/products/ps12806/tsd_products_support_series_home.html

Release Notes

The release notes are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-6000-series-switches/products-release-notes-list.html>

Configuration Guides

These guides are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-6000-series-switches/products-installation-and-configuration-guides-list.html>

The documents in this category include:

- *Cisco Nexus 6000 Series NX-OS Adapter-FEX Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS FabricPath Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS FCoE Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Fundamentals Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Interfaces Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Layer 2 Switching Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Multicast Routing Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Quality of Service Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS SAN Switching Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Security Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS System Management Configuration Guide*
- *Cisco Nexus 6000 Series NX-OS Unicast Routing Configuration Guide*

Installation and Upgrade Guides

These guides are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-6000-series-switches/products-installation-guides-list.html>

The document in this category include:

- *Cisco Nexus 6000 Series NX-OS Software Upgrade and Downgrade Guides*

Licensing Guide

The *License and Copyright Information for Cisco NX-OS Software* is available at http://www.cisco.com/en/US/docs/switches/datacenter/sw/4_0/nx-os/license_agreement/nx-oss_w_lisns.html.

Command References

These guides are available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-6000-series-switches/products-command-reference-list.html>

The documents in this category include:

- *Cisco Nexus 6000 Series NX-OS Fabric Extender Command Reference*
- *Cisco Nexus 6000 Series NX-OS FabricPath Command Reference*
- *Cisco Nexus 6000 Series NX-OS Fundamentals Command Reference*
- *Cisco Nexus 6000 Series NX-OS Interfaces Command Reference*
- *Cisco Nexus 6000 Series NX-OS Layer 2 Interfaces Command Reference*
- *Cisco Nexus 6000 Series NX-OS Multicast Routing Command Reference*
- *Cisco Nexus 6000 Series NX-OS Quality of Service Command Reference*

- *Cisco Nexus 6000 Series NX-OS Security Command Reference*
- *Cisco Nexus 6000 Series NX-OS System Management Command Reference*
- *Cisco Nexus 6000 Series NX-OS TrustSec Command Reference*
- *Cisco Nexus 6000 Series NX-OS Unicast Routing Command Reference*
- *Cisco Nexus 6000 Series NX-OS Virtual Port Channel Command Reference*

Technical References

The *Cisco Nexus 6000 Series NX-OS MIB Reference* is available at http://www.cisco.com/en/US/docs/switches/datacenter/nexus6000/sw/mib/reference/NX6000_MIBRef.html.

Error and System Messages

The *Cisco Nexus 6000 Series NX-OS System Message Guide* is available at http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus6000/sw/system_messages/reference/sl_nxos_book.html.

Troubleshooting Guide

The *Cisco Nexus 6000 Series NX-OS Troubleshooting Guide* is available at <http://www.cisco.com/c/en/us/support/switches/nexus-6000-series-switches/tsd-products-support-troubleshoot-and-alerts.html>.

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to: nexus5k-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](#).

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 5000 and 6000 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 5000 and 6000 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)N1(1)	NX-API, on page 9
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 21
XML Management Interface	This feature was modified to support NETCONF Enhancements	7.3(0)N1(1)	XML Management Interface, on page 35

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)N1(1)	Open Agent Container, on page 59
Using Chef client with Cisco NX-OS	Support for Chef client on Cisco NX-OS introduced.	7.3(0)N1(1)	Using Chef Client with Cisco NX-OS, on page 65
Using Puppet agent with Cisco NX-OS	Support for Puppet agent introduced.	7.3(0)N1(1)	Using Puppet Agent with Cisco NX-OS, on page 69



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 5000 and 6000 Series devices is as follows:

- **Resilient**
Provides critical business-class availability.
- **Modular**
Has extensions that accommodate business needs.
- **Highly Programmatic**
Allows for rapid automation and orchestration through Application Programming Interfaces (APIs).
- **Secure**
Protects and preserves data and operations.
- **Flexible**
Integrates and enables new technologies.
- **Scalable**
Accommodates and grows with the business and its requirements.
- **Easy to use**
Reduces the amount of learning required, simplifies deployment, and provides ease of manageability.

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

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

Standard Network Manageability Features

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

Advanced Automation Feature

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

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

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

PowerOn Auto Provisioning Support

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

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

For more details about POAP, see the *Cisco Nexus 5000 Series NX-OS Fundamentals Configuration Guide*.

OpenStack Integration

The Cisco Nexus 5000 Series and 6000 Series devices support the Cisco Nexus plugin for OpenStack Networking, also known as Neutron (<http://www.cisco.com/web/solutions/openstack/index.html>). The plugin

allows you to build an infrastructure as a service (IaaS) network and to deploy a cloud network. With OpenStack, you can build an on-demand, self-service, multitenant computing infrastructure. However, implementing OpenStack's VLAN networking model across virtual and physical infrastructures can be difficult.

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

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

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

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

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

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

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

Programmability Support

Cisco NX-OS on Cisco Nexus 5000 and 6000 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 5000 Series and 6000 Series platforms. This support is delivered by NX-API, an open source webservice. 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 5000 Series and 6000 Series platforms.

Python Scripting

Cisco Nexus 5000 Series and 6000 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 5000 Series and 6000 Series devices support for Python, see the *Cisco Nexus 5000 Series NX-OS Fundamentals Configuration Guide*, *Cisco Nexus 6000 Series NX-OS Fundamentals Configuration Guide*.

Tcl Scripting

Cisco Nexus 5000 Series and 6000 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.



NX-API

- [About NX-API, page 9](#)
- [Using NX-API, page 10](#)
- [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 5000 and 6000 Series devices. NX-API supports **show** commands and configurations.

NX-API supports JSON-RPC.

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. The Nginx process, and all of its children processes, are under Linux cgroup protection where the CPU and memory usage is capped. If the Nginx memory usage exceeds the cgroup limitations, the Nginx process is restarted and restored.

Message Format

NX-API is an enhancement to the Cisco Nexus 5000 and 6000 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 5000 and 6000 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 or JSON output format.

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 enable NX-API:

- Enable the management interface.

```
switch# configure terminal
switch(config)# interface mgmt 0
```



```
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context management
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

- Enable the NX-API `nxapi` feature.

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

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

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
  <version>0.1</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>session1</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</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": "0.1",
    "type": "cli_show",
    "chunk": "0",
    "sid": "session1",
    "input": "show switchname",
    "output_format": "json"
  }
}
```

Response:

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "0.1",
    "sid": "eoc",
    "outputs": {
      "output": {
        "body": {
          "hostname": "switch"
        },
        "input": "show switchname",
        "msg": "Success",
        "code": "200"
      }
    }
  }
}
```

```

    }
  }
}

```

Sample NX-API Scripts

The sample scripts demonstrate how a script is used with NX-API. The scripts are available at <https://github.com/datacenter/nxos/tree/master/nxapi/samples>.

- Cable Checker (check_cable.py)
- Cable Checker Blueprint (connectivity.json)

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 5000 and 6000 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 management
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

The screenshot displays the NX-API Developer Sandbox interface. At the top, the Cisco logo and the title "NX-API Developer Sandbox" are visible. Below the title is a large text input area containing the command "show ver". Underneath the input area are two buttons: "POST" (blue) and "Reset" (orange). Below the input area, there are two panels: "REQUEST:" and "RESPONSE:". The "REQUEST:" panel shows a JSON object with the following structure:

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

The "RESPONSE:" panel shows a JSON object with the following structure:

```
{
  "jsonrpc": "2.0",
  "result": {
    "body": {
      "header_str": "Cisco Nexus Operating System
tac\nCopyright (C) 2002-2014, Cisco and/or its affi
rtain works contained in this software are\nowned b
r their own\nlicenses, such as open source. This s
e stated, there is no warranty, express or implied,
ntability and fitness for a particular purpose.\nC
r\nthe GNU General Public License (GPL) version 2.6
or the GNU\nLesser General Public License (LGPL) V
) Version 2.0. \nA copy of each such license is ava
.0.php and\nhttp://opensource.org/licenses/gpl-3.0
.1.php and\nhttp://www.gnu.org/licenses/old-licens
"bios_ver_str": "07.06",
"kickstart_ver_str": "6.1(2)I2(2a)",
"bios_cpl_time": "03/01/2014",
"kick_file_name": "bootflash:///n9000-dk9.6.1
"kick_cpl_time": " 6/3/2014 13:00:00",
```

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 <i>port</i>	Specifies a port.
no nxapi {http https}	Disables HTTP/HTTPS.
show nxapi	Displays port information.
nxapi certificate <i>certpath</i> key <i>keypath</i>	Specifies the upload of the following: <ul style="list-style-type: none"> • HTTPS certificate when <i>certpath</i> is specified. • HTTPS key when <i>keypath</i> is specified.
nxapi certificate <i>enable</i>	Enables a certificate.

NX-API Request Elements

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

You 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
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> • 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. <p>Note</p> <ul style="list-style-type: none"> • 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.

NX-API Request Element	Description				
<p><i>chunk</i></p>	<p>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.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="787 464 1479 590"> <tr> <td data-bbox="787 464 899 527">0</td> <td data-bbox="899 464 1479 527">Do not chunk output.</td> </tr> <tr> <td data-bbox="787 527 899 590">1</td> <td data-bbox="899 527 1479 590">Chunk output.</td> </tr> </table> <p>Note Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.</p> <p>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 &lt; ; > is converted into &gt;).</p> <p>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.</p>	0	Do not chunk output.	1	Chunk output.
0	Do not chunk output.				
1	Chunk output.				
<p><i>sid</i></p>	<p>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 <i>sid</i> to match the <i>sid</i> of the previous response message.</p>				
<p><i>input</i></p>	<p>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.</p> <p>Note Multiple commands are separated with ";". (The ; must be surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="787 1409 1479 1566"> <tr> <td data-bbox="787 1409 911 1482">cli_show</td> <td data-bbox="911 1409 1479 1482">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="787 1482 911 1566">cli_conf</td> <td data-bbox="911 1482 1479 1566">interface Eth4/1 ; no shut ; switchport</td> </tr> </table>	cli_show	show version ; show interface brief ; show vlan	cli_conf	interface Eth4/1 ; no shut ; switchport
cli_show	show version ; show interface brief ; show vlan				
cli_conf	interface Eth4/1 ; no shut ; switchport				

NX-API Request Element	Description				
<i>output_format</i>	<p>The available output message formats are the following:</p> <table border="1" data-bbox="824 317 1518 449"> <tbody> <tr> <td data-bbox="824 317 1060 380">xml</td> <td data-bbox="1060 317 1518 380">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="824 380 1060 449">json</td> <td data-bbox="1060 380 1518 449">Specifies output in JSON format.</td> </tr> </tbody> </table> <p>Note The Cisco Nexus 5000 and 6000 Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.</p> <p>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.</p> <p>The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).</p>	xml	Specifies output in XML format.	json	Specifies output in JSON format.
xml	Specifies output in XML format.				
json	Specifies output in JSON format.				

NX-API Response Elements

The NX-API elements that respond to a CLI command are listed in the following table:

Table 5: 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	<p>Tag that encloses all command outputs.</p> <p>When multiple commands are in <code>cli_show</code> or <code>cli_show_ascii</code>, each command output is enclosed by a single output tag.</p> <p>When the message type is <code>cli_conf</code> or <code>bash</code>, there is a single output tag for all the commands because <code>cli_conf</code> and <code>bash</code> commands require context.</p>
output	<p>Tag that encloses the output of a single command output.</p> <p>For <code>cli_conf</code> and <code>bash</code> message types, this element contains the outputs of all the commands.</p>

NX-API Response Element	Description
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.

Command Types

There are a number of command “types” available for sending show and configuration commands to a device using the NX-API. The type you need to use depends on the actual command you want to send: `cli_show`. You should use the “`cli_show`” type when you want to send a show command that supports structured XML output. To know whether a command supports XML output or not, you can go to the CLI of the switch and run the command with the “`| xml`” option on the end—you will be able to see whether that command returns XML output or not. If you try and use the “`cli_show`” command type with a command that does not support XML, you will receive a message stating “structured output unsupported” from the API:

```
<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <input>show clock</input>
      <msg>Structured output unsupported</msg>
      <code>501</code>
    </output>
  </outputs>
</ins_api>
```

If that happens, you will need to use the command type— “`cli_show_ascii`”.

- **cli_show_ascii**—This command type returns output in ASCII format with the entire output inside one `<body>` element. Any command on the switch (including the ones that do not support structured XML output) should work with this command type, although it will be more difficult to parse compared to those commands that return XML.
- **cli_conf**— Use this command type when you want to send configuration commands (as opposed to show commands) to the API.

Additional References

This section provides additional information related to implementing NX-API.

- [NX-API DevNet Community](#)
- [NX-API Github \(NX-OS Programmability scripts\)](#)



Python API

- [Information About the Python API](#), page 21
- [Installing Python](#), page 22
- [Using Python](#), page 22
- [Application Programming Interface \(API\) Functions](#), page 23

Information 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.

**Note**

You can know the latest development on Cisco Nexus 5000 Series switches and contribute your ideas by joining the community code sharing page: <https://github.com/datacenter>

The Cisco Nexus 5000 series switches with Releases 5.2(1)N1(1) and later and the Cisco Nexus 6000 series switches with Releases 6.0(2)N1(1) and later, support all the features available in Python v2.7.2.

The Python scripting capability on Cisco Nexus 5000 and Cisco Nexus 6000 series switches enables you to perform the following tasks:

- Run a script to verify configuration on switch bootup.
- Back up a configuration.
- Proactive congestion management by monitoring and responding to buffer utilization characteristics.
- Integration with the Power-On Auto Provisioning or EEM modules.

- Ability to perform a job at a specific time interval (such as Port Auto Description).
- Programmatic access to the switch command line interface (CLI) to perform various tasks.

Installing Python

The Python interpreter is available by default on the Cisco NX-OS software. You can invoke Python by entering the `python` command, and write scripts to access Cisco NX-OS APIs by importing the `cisco.py` module using the `import cisco` command.

Installing Third Party Pure Python Packages

You can install the third party pure Python package by copying `mypkg.tgz` on your server. Perform the following steps to extract and install the third party package:

- Secure copy the tar file by executing the `copy scp://user@server/path/to/mypkg.tgz bootflash:mypkg.tgz` vrf management command
- Untar the `mypkg.tgz` file by using the `tar extract bootflash:mypkg.tgz` command.
- Move the extracted file to bootflash by using the `move bootflash:mypkg-1.2/* bootflash:` command.
- You can install the package by using the `python setup.py install` command.
- Remove the copied file from bootflash.
- You can use the third party package in scripts or in the Python shell.

```
switch# python
>>> import mypkg
```



Note

You will be able to install the third party packages using the `easy_install` command, in the future releases.

Using Python

This section describes how to write and execute Python scripts by passing parameters and includes the following topics:

Entering Python Shell

You can enter the Python shell by using the `python` command without any parameters.

```
switch# python
Python 2.7.2 (default, Oct 11 2011, 13:55:49)
[GCC 3.4.3 (MontaVista 3.4.3-25.0.143.0800417 2008-02-22)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NX-OS lib!
>>> print 'helo world!'
helo world!
```

```
>>>exit()
switch#
```

Executing Scripts

For NX-OS releases earlier than 7.0N, you can execute a Python script by using the **python** *<filename>* command.

```
switch# python test.py
['/bootflash/test.py']
doing 0/1
doing 0/2
doing 1/2
switch#
```

For NX-OS releases 7.0N and later, you can execute a Python script by using the **source** *<filename>* command.

```
switch# source test.py
['/bootflash/test.py']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Passing Parameters to the Script

For NX-OS releases earlier than 7.0N, you can pass parameters to a Python script by using the **python** *<filename>* [*arg1, arg2, arg3,.....*] command.

```
switch# python test.py foo bar 1 2
['/bootflash/test.py', 'foo', 'bar', '1', '2']
doing 0/1
doing 0/2
doing 1/2
switch#
```

For NX-OS releases 7.0N and later, you can pass parameters to a Python script by using the **source** *<filename>* [*arg1, arg2, arg3,.....*] command.

```
switch# source test.py foo bar 1 2
['/bootflash/test.py', 'foo', 'bar', '1', '2']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Application Programming Interface (API) Functions

This chapter provides information about the following Python Application Programming Interface (API) functions. This chapter includes the following sections:

Routes()

Synopsis

`Routes()` - Class Object

Syntax

`Routes()`

Description

Instantiates an object of the Routes class.

Parameters

None.

Returns

An object of Routes class.

Example

```
rObj = Routes()
```

show_arp_table()

Synopsis

`show_arp_table()`

Syntax

`Routes.show_arp_table()`

Description

Executes the show ip arp command and returns the output.

Parameters

None.

Returns

Returns the ARP table entries on the switch.

Example

```
routeObj = Routes()
data = routeObj.show_arp_table().get_output()
```

Sample Output

```
Flags: D - Static Adjacencies attached to down interface
IP ARP Table for context default
Total number of entries: 4
Address      Age      MAC Address  Interface
50.1.201.2   00:02:10  547f.ee40.5a7c  Vlan201
50.1.1.10    00:07:53  547f.ee62.f801  Ethernet1/34
50.1.2.10    00:08:31  547f.ee62.f801  Ethernet1/35
50.1.3.10    00:08:31  547f.ee62.f801  Ethernet1/35.1
<cisco.CLI object at 0xb7c1462c>
```

show_vsh_routes()

Synopsis

```
show_vsh_routes()
```

Syntax

```
Routes.show_vsh_routes()
```

Description

Executes the show ip fib route and returns the output.

Parameters

None.

Returns

Returns the software route entries.

Example

```
routeObj = Routes()
data = routeObj.show_vsh_routes().get_output()
```

Sample Output

```
IPv4 routes for table default/base
-----+-----+-----
Prefix      | Next-hop      | Interface
-----+-----+-----
0.0.0.0/32   | Drop          | Null0
50.1.1.0/24  | Attached      | Ethernet1/34
50.1.1.0/32  | Drop          | Null0
50.1.1.10/32 | 50.1.1.10     | Ethernet1/34
50.1.1.100/32 | Receive       | sup-eth1
50.1.1.255/32 | Attached      | Ethernet1/34
50.1.2.0/24  | Attached      | Ethernet1/35
```

```

50.1.2.0/32          Drop          Null0
50.1.2.10/32       50.1.2.10   Ethernet1/35
50.1.2.100/32      Receive      sup-eth1
50.1.2.255/32      Attached     Ethernet1/35
50.1.3.0/24        Attached     Ethernet1/35.1
50.1.3.0/32        Drop         Null0
50.1.3.10/32       50.1.3.10   Ethernet1/35.1
50.1.3.100/32     Receive      sup-eth1
50.1.3.255/32     Attached     Ethernet1/35.1
<cisco.CLI object at 0xb7b0a6ac>

```

show_hw_routes()

Synopsis

```
show_hw_routes()
```

Syntax

```
Routes.show_hw_routes()
```

Description

Computes the hardware routes and returns the output.

Parameters

None.

Returns

Returns the hardware route entries.

Example

```
routeObj = Routes()
data = routeObj.show_hw_routes()
```

Sample Output

```

-----+-----+-----
Prefix          | Next-hop          | Interface
-----+-----+-----
50.1.1.100/32   | Receive          | sup-eth1
50.1.2.100/32   | Receive          | sup-eth1
50.1.201.1/32   | Receive          | sup-eth1
0.0.0.0/32      | Drop             | Null0
50.1.3.0/32     | Drop             | Null0
50.1.201.0/32   | Drop             | Null0
50.1.2.255/32   | Attached         | sup-hi
50.1.1.255/32   | Attached         | sup-hi
60.1.1.0/32     | Drop             | Null0
50.1.3.255/32   | Attached         | sup-hi
50.1.201.255/32 | Attached         | sup-hi
255.255.255.255/32 | Receive         | sup-eth1

```


verify_routes()

Synopsis

```
verify_routes()
```

Syntax

```
Routes.verify_routes()
```

Description

Verifies the software and hardware routes.

Parameters

None.

Returns

Returns the number of routes matched and unmatched between hardware and software.

Example

```
routeObj = Routes()  
found,nFound = routeObj.verify_routes()
```

Sample Output

```
Routes verified and found: 26  
Routes not found:  
50.1.205.0/24      3  
51.1.1.0/24       3  
51.1.2.0/24       4  
51.1.3.0/24       6  
100.1.1.0/24      7  
100.1.2.0/24      7  
100.1.3.0/24      7  
101.1.1.0/24      7  
101.1.2.0/24      7  
101.1.3.0/24      7  
120.1.1.0/24      7
```

verify_arp_table()

Synopsis

```
verify_arp_table()
```

Syntax

```
Routes.verify_arp_table()
```

Description

Verifies the software and hardware ARP table entries.

Parameters

None.

Returns

Returns the number of ARP table entries matched and unmatched between hardware and software.

Example

```
routeObj = Routes()
found,notfound = routeObj.verify_arp_table()
```

Sample Output

```
Flags: D - Static Adjacencies attached to down interface
IP ARP Table for context default
Total number of entries: 4
Address          Age          MAC Address   Interface
50.1.201.2       00:02:31    547f.ee40.5a7c  Vlan201
50.1.1.10        00:08:15    547f.ee62.f801  Ethernet1/34
50.1.2.10        00:08:53    547f.ee62.f801  Ethernet1/35
50.1.3.10        00:08:53    547f.ee62.f801  Ethernet1/35.1
mac address:54:7f:ee:40:5a:7c
Arp entry for 50.1.201.2 547f.ee40.5a7c Vlan201 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.1.10 547f.ee62.f801 Ethernet1/34 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.2.10 547f.ee62.f801 Ethernet1/35 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.3.10 547f.ee62.f801 Ethernet1/35.1 found in HW
```

CheckPortDiscards()

Synopsis

```
CheckPortDiscards(<port>)
```

Syntax

```
CheckPortDiscards('ethernet1/1')
```

Description

Check the input discards for given port. If discard is more than 0, query and print the discard reason from broadcom.

Parameters

port

Returns

None.

Example

```
c = CheckPortDiscards('eth1/1')
```

Sample Output

```
Ethernet1/1 is up
Hardware: 100/1000/10000 Ethernet, address: 547f.ee57.dd28 (bia 547f.ee57.dd28)
MTU 1500 bytes, BW 10000000 Kbit, DLY 10 usec,
    reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA
Port mode is trunk
full-duplex, 10 Gb/s, media type is 10G
Beacon is turned off
Input flow-control is off, output flow-control is off
Rate mode is dedicated
Switchport monitor is off
EtherType is 0x8100
Last link flapped 00:42:16
Last clearing of "show interface" counters never
30 seconds input rate 5016 bits/sec, 627 bytes/sec, 6 packets/sec
30 seconds output rate 3232 bits/sec, 404 bytes/sec, 5 packets/sec
Load-Interval #2: 5 minute (300 seconds)
    input rate 4.69 Kbps, 7 pps; output rate 2.82 Kbps, 4 pps
RX
 297 unicast packets 20588 multicast packets 5 broadcast packets
20890 input packets 1848701 bytes
0 jumbo packets 0 storm suppression packets
0 giants 0 input error 0 short frame 0 overrun 0 underrun
0 watchdog 0 if down drop
0 input with dribble 0 input discard(includes ACL drops)
0 Rx pause
TX
 262 unicast packets 16151 multicast packets 5 broadcast packets
16418 output packets 1407200 bytes
0 jumbo packets
0 output errors 0 collision 0 deferred 0 late collision
0 lost carrier 0 no carrier 0 babble
0 Tx pause
2 interface resets
zero discards
```

transfer()

Synopsis

```
transfer()
```

Syntax

```
transfer (<protocol>, <host>, <source>, <dest>, <vrf>, <login_timeout>, <user>, <password>)
```

Description

API to transfer file specified in <source> from <host> to the path mentioned in <dest> using <protocol>. Protocol can be scp, tftp, ftp or sftp.

Parameters

protocol, host, source, dest, vrf, login_timeout, user, password.

Returns

Returns True if transfer was successful.

Example

Transfer using scp:

```
c = transfer("scp", "10.193.190.100", "/tftpboot/transfer_test_image", "transfer_test_image",
            user="scpUser", password="scpPasswd")
```

Transfer using sftp:

```
c = transfer("sftp", "10.193.190.100", "/tftpboot/transfer_test_image", "transfer_test_image",
            user="sftpUser", password="sftpPasswd")
```

Transfer using tftp:

```
c = transfer("tftp", "10.193.190.100", "/transfer_test_image", "transfer_test_image",
            user="", password="")
```

Transfer using ftp:

```
c = transfer("ftp", "10.193.190.51", "golden/home/su-ash/transfer_test_image",
            "transfer_test_image", user="ftpUser", password="ftpPasswd")
```

CLI()

Synopsis

CLI() - Class Object

Syntax

CLI (<command>, <do_print>)

Description

Instantiates an object of the CLI class with the CLI command specified in <command>. <do_print> when set to False does not print the output of the command and prints the output when set to True, which is the default.

**Note**

The CLI command name is case sensitive (uppercase).

Parameters

command, do_print

Returns

An object of CLI class.

Example

```
c = CLI ('show runn inter eth1/1')
```

Sample Output

```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:33:24 2012
version 5.0(3)U3(1)
interface Ethernet1/1
  switchport mode trunk
  uddl enable
  channel-group 12
<cisco.CLI object at 0xb7ae948c>
```

get_output()

Synopsis

```
get_output()
```

Syntax

```
CLI.get_output()
```

Description

Returns the output of the CLI command.

Parameters

None.

Returns

Output of the CLI command.

Example

```
c = CLI ('show runn inter eth1/1')
c.get_output()
```

Sample Output

```
['', '!Command: show running-config interface Ethernet1/1', '!Time: Mon Feb 27 14:36:10', '2012', '', 'version 5.0(3)U3(1)', '', 'interface Ethernet1/1', ' switchport mode trunk', ' uddl enable', ' channel-group 12', '', '']
```

rerun()

Synopsis

```
rerun()
```

Syntax

```
CLI.rerun()
```

Description

Reruns the command.

Parameters

None.

Returns

None.

Example

```
c = CLI ('show runn inter eth1/1')
c.rerun()
```

Sample Output

```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:37:05 2012
version 5.0(3)U3(1)
interface Ethernet1/1
  switchport mode trunk
  udld enable
  channel-group 12
```

History()

Synopsis

History() - Class Object

Syntax

```
History()
```

Description

Instantiates an object of the History class.

Parameters

None.

Returns

An object of History class.

Example

```
a = History()
```

get_history()

Synopsis

```
get_history()
```

Syntax

```
History.get_history()
```

Description

Gets the history of CLI commands executed so far.

Parameters

None.

Returns

Returns the history of commands executed.

Example

```
a = History()
a.get_history()
```

clear_history()

Synopsis

```
clear_history()
```

Syntax

```
History.clear_history()
```

Description

Clears history.

Parameters

None.

Returns

None.

Example

```
a = History()
a.clear_history()
```




CHAPTER 5

XML Management Interface

- [XML Management Interface, page 35](#)
- [Feature History for XML Management Interface, page 35](#)
- [About the XML Management Interface, page 36](#)
- [Licensing Requirements for the XML Management Interface, page 37](#)
- [Prerequisites to Using the XML Management Interface, page 38](#)
- [Using the XML Management Interface, page 38](#)
- [Example XML Instances, page 50](#)
- [Additional References, page 57](#)

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 6: Feature History XML Management Interface

Feature Name	Releases	Feature Information
XML Management Interface	7.3(0)N1(1)	<p>Added the following NetConf enhancements:</p> <ul style="list-style-type: none"> • 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 41](#) 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 7: NETCONF Layers

Layer	Example
Transport protocol	SSHv2

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

The following is a description of the four NETCONF layers:

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

SSH xmlagent

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



Note

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

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

The XML schemas that define XML configuration instances that you can use are described in [Creating NETCONF XML Instances](#), on page 41 .

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 <i>Cisco NX-OS Licensing Guide</i> .

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.



Note

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

Configuring the SSH and the XML Server Options Through the CLI

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

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



Note

The XML server timeout applies only to active sessions.

For more information about configuring SSH, see

http://www.cisco.com/itd/cs/wish/taurus/nx-ssh-security/configuring/index_Cisco_Nexus_7000_NX-OS_Security_Configuration_Guide_Release_7.html

For more information about the XML commands, see

http://www.cisco.com/itd/cs/wish/taurus/nx-xml-management/index_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide_Release_7.html

-
- 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.

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 an 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 the CLI:

```
switch# configure terminal
switch(config)# xml server validate all
switch(config)# xml server terminate 8665
switch(config)# no feature ssh
switch(config)# feature ssh server
switch(config)# xml server max-session 6
switch(config)# xml server timeout 2400
switch(config)# copy running-config startup-config
```

Starting an SSHv2 Session

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

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

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



Note

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

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

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

Sending a Hello Message

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

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


Note

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

Hello Message from a Server

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

Hello Message from a Client

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

Obtaining XML Schema Definition (XSD) Files

-
- Step 1** From your browser, navigate to the Cisco software download site at <http://software.cisco.com/download/navigator.html>
- The Download Software window opens.
- Step 2** From the list of products displayed, choose **Switches > Data Center Switches > platform > model** .
- Step 3** If you are not already logged in as a registered Cisco user, you are prompted to log in now.
- Step 4** From the **Select a Software Type** list, choose **NX-OS XML Schema Definition**.
- Step 5** Find the desired release and click **Download**.
- Step 6** If you are requested to, follow the instructions to apply for eligibility to download strong encryption software images. The Cisco End User License Agreement is displayed.
- Step 7** Click **Agree** and follow the instructions to download the file to your PC.
-

Sending an XML Document to the XML Server

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

Following are the guidelines to send the XML document to the XML Server:

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

Creating NETCONF XML Instances

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

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

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

Table 8: 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"
  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 `<rpc>` element has a `message-id` attribute. This `message-id` attribute is replicated in the `<rpc-reply>` and can be used to correlate requests and replies. The `<rpc>` node also contains the following XML namespace declarations:

- NETCONF namespace declaration—The `<rpc>` and NETCONF tags that are defined in the "urn:ietf:params:xml:ns:netconf:base:1.0" namespace, are present in the `netconf.xsd` schema file.
- Device namespace declaration—Device tags encapsulated by the `<rpc>` and NETCONF tags are defined in other namespaces. Device namespaces are feature oriented. Cisco NX-OS feature tags are defined in different namespaces. *RPC Request Tag <rpc>* is an example that uses the `nfcli` feature. It declares that the device namespace is "xmlns=http://www.cisco.com/nxos:1.0:nfcli". `nfcli.xsd` contains this namespace definition. For more information, see [Obtaining XML Schema Definition \(XSD\) Files](#), on page 40.

RPC Request Tag `<rpc>`

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

Configuration Request

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

Note:

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

NETCONF Operations Tags

NETCONF provides the following configuration operations:

Table 9: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 51
commit	Sets the running configuration to current contents of candidate configuration.	NETCONF Commit Instance - Candidate Configuration Capability, on page 55
confirmed-commit	Provides the parameters to commit the configuration for a specified period of time. If this operation is not followed by a commit operation within the confirm-timeout period, the configuration will be reverted to the state prior to the confirmed-commit operation.	NETCONF Confirmed Commit Instance, on page 55
copy-config	Copies the content of source configuration datastore to the target datastore.	NETCONF Copy Config Instance, on page 52
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 52 NETCONF Rollback-On-Error Instance, on page 56
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 41
get-config	Retrieves all or part of a configuration	NETCONF Get Config Instance, on page 53
kill-session	Closes the specified XML server session. Note that you cannot close your own session.	NETCONF Kill Session Instance, on page 51
lock	Allows the client to lock the configuration system of a device.	NETCONF Lock Instance, on page 54

NETCONF Operation	Description	Example
unlock	Releases the configuration lock issued by the session earlier.	NETCONF Unlock Instance, on page 55
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	NETCONF Validate Capability Instance, on page 56

Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See [Obtaining XML Schema Definition \(XSD\) Files, on page 40](#).

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 the NETCONF instances ([Creating NETCONF XML Instances, on page 41](#)) is shown in the following examples.

Show Xml Device Tags

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

Server Status Device Tags

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

Device Tag Response

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



Note

"__XML__OPT_Cmd_show_xml__readonly__" is optional. This tag represents the response. For more information on responses, see [RPC Response Tag](#), on page 49.

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

The following example shows you that the namespace defining tag used to execute operations on this device is http://www.cisco.com/nxos:1.0:nfcli and that the nfcli.xsd file can be used to build requests.

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

XML Example

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

```
</nf:rpc-reply>
]]>]]>
```

Extended NETCONF Operations

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

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

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

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

Table 10: Tags

Tag	Description
<code><exec-command></code>	Executes a CLI command.
<code><cmd></code>	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 <code><cmd></code> tags as part of the same request. For more information, see the Example on <i>Configuration CLI Commands Sent Through <code><exec-command></code></i> .

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

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>]]>]]>
```

Response to CLI Commands Sent Through <exec-command>

The following is the response to the send operation:

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

Show CLI Commands Sent Through <exec-command>

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

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

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

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

Failed Configuration

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 1.1.1.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

As a result of a command's 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 **config** and **show** commands combined in the same instance.

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>
]]>]]>
```

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 a client, the XML server sends an XML response enclosed in the RPC response tag `<rpc-reply>`.

RPC Response Tag

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

RPC Response Tag `<rpc-reply>`

```

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

```

RPC Response Elements

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

Table 11: RPC Response Elements

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

Interpreting the Tags Encapsulated in the Data Tag

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

RPC Reply Data

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



Note

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

Example XML Instances

This section provides examples of the following XML instances:

- [NETCONF Close Session Instance](#), on page 51
- [NETCONF Kill Session Instance](#), on page 51
- [NETCONF Copy Config Instance](#), on page 52

- [NETCONF Edit Config Instance](#), on page 52
- [NETCONF Get Config Instance](#), on page 53
- [NETCONF Lock Instance](#), on page 54
- [NETCONF Unlock Instance](#), on page 55
- [NETCONF Commit Instance - Candidate Configuration Capability](#), on page 55
- [NETCONF Confirmed Commit Instance](#), on page 55
- [NETCONF Rollback-On-Error Instance](#), on page 56
- [NETCONF Validate Capability Instance](#), on page 56

NETCONF Close Session Instance

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

Close Session Request

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

Close-session Response

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

NETCONF Kill Session Instance

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

Kill Session Request

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

Kill Session Response

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

NETCONF Copy Config Instance

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

Copy Config Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
<target>
<running/>
</target>
<source>
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

Copy Config Response

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

NETCONF Edit Config Instance

The following examples show the use of NETCONF edit config:

Edit Config Request

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

Edit Config Response

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

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

- create
- merge
- delete

Edit Config: Delete Operation Request

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

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

Response to Edit Config: Delete Operation

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

NETCONF Get Config Instance

The following examples show the use of the NETCONF get-config:

Get Config Request to Retrieve the Entire Subtree

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

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

Get Config Response with Results of 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 examples show a lock request, a success response and a response to an unsuccessful attempt:

Lock Request

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

Response to a Successful Acquisition of Lock

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

Response to an Unsuccessful Attempt to Acquire 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 examples show the use of NETCONF unlock operation:

Unlock Request

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

Response to an Unlock Request

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

NETCONF Commit Instance - Candidate Configuration Capability

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

Commit Operation

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

Commit Reply

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

NETCONF Confirmed Commit Instance

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

Confirmed Commit Request

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

Confirmed Commit Response

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

NETCONF Rollback-On-Error Instance

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

Rollback-On-Error Capability

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

Rollback-On-Error Response

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

NETCONF Validate Capability Instance

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

Validate Request

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

Response to Validate Request

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

Additional References

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

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)



Open Agent Container

- [Open Agent Container, page 59](#)

Open Agent Container

This chapter explains the Open Agent Container (OAC) environment and its installation in the following Cisco Nexus Switches:

- Cisco Nexus 5600 Switches
- Cisco Nexus 6000 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 12: Feature History for Open Agent Container

Feature Name	Releases	Feature Information
Open Agent Container (OAC)	7.3(0)N1(1)	<p>This feature was introduced in the Cisco Nexus 5600 Switches and the Cisco Nexus 6000 series switches.</p> <p>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.</p>

About Open Agent Container

From Cisco NX-OS 7.3(0)N1(1) and later releases, Cisco Nexus 5600 Switches and Cisco Nexus 6000 Series 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 managementcmd** 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** *virtual-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
```



Using Chef Client with Cisco NX-OS

- [Using Chef Client with Cisco NX-OS](#), page 65

Using Chef Client with Cisco NX-OS

Feature History for Chef Support

This table lists the release history for this feature.

Table 13: Feature History for Chef Support

Feature Name	Releases	Feature Information
Support for Chef	7.3(0)N1(1)	This feature was introduced in Cisco Nexus 5600 and Cisco Nexus 6000 series 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 5600 Series switch
 - Cisco Nexus 6000 Series switch
 - Cisco NX-OS release 7.3(0)N1(1) or later for Cisco Nexus 5600 and Cisco Nexus 6000 series switches.
- Chef agents cannot run natively on Cisco Nexus 5600 and Cisco Nexus 6000 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 59](#)
- 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 59](#).

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



Using Puppet Agent with Cisco NX-OS

- [Using Puppet Agent with Cisco NX-OS, page 69](#)

Using Puppet Agent with Cisco NX-OS

Feature History for Puppet Support

This table lists the release history for this feature.

Table 14: Feature History for Puppet Support

Feature Name	Releases	Feature Information
Support for Puppet	7.3(0)N1(1)	This feature was introduced in Cisco Nexus 5600 and Cisco Nexus 6000 series 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 5600 Series switch.
 - Cisco Nexus 6000 Series switch.
 - Cisco NX-OS release 7.3(0)N1(1) or later for Cisco Nexus 5600 and Cisco Nexus 6000 series switches.
- Puppet agents cannot run natively on Cisco Nexus 5600 and Cisco Nexus 6000. 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 59.

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/



NX-API Response Codes

- [Table of NX-API Response Codes, page 73](#)

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 15: 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.

