Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 6.x

First Published: 2013-11-20

Last Modified: 2019-09-21

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

CONTENTS

Preface

This preface includes the following sections:

Audience

This publication is for network administrators who install, configure, and maintain Cisco Nexus switches.

Document Conventions

Command descriptions use the following conventions:

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

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

Related Documentation for Cisco Nexus 9000 Series Switches

The entire Cisco Nexus 9000 Series switch documentation set is available at the following URL:

http://www.cisco.com/en/US/products/ps13386/tsd_products_support_series_home.html

Documentation Feedback

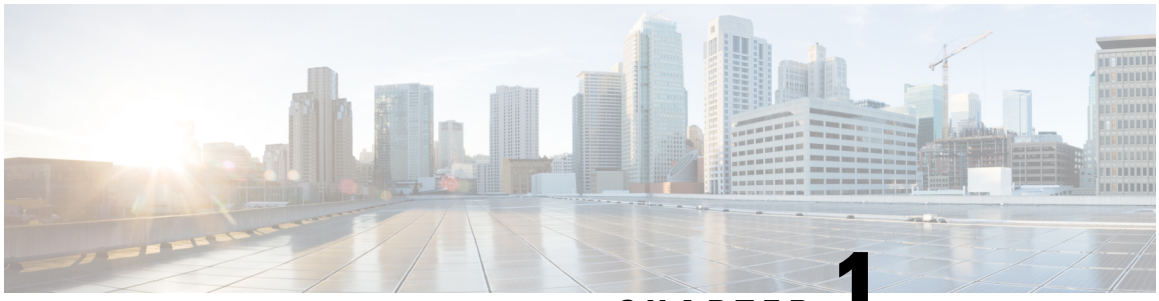
To provide technical feedback on this document, or to report an error or omission, please send your comments to nexus9k-docfeedback@cisco.com. We appreciate your feedback.

Communications, Services, and Additional Information

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

Cisco Bug Search Tool

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



CHAPTER 1

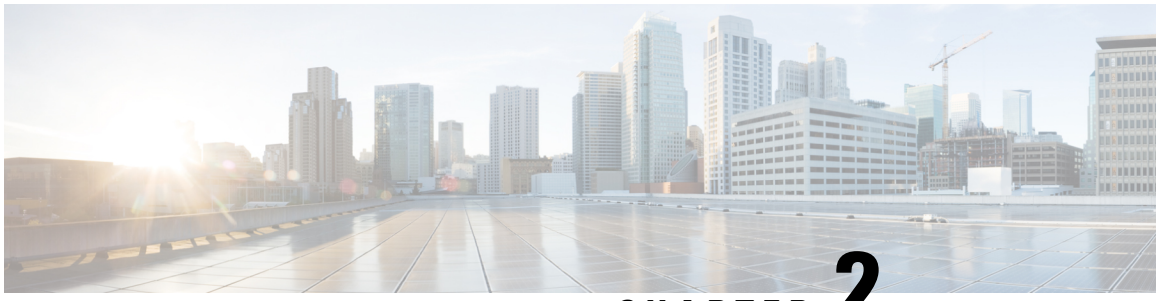
New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 6.x*.

New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 6.x*.

New and Changed Information



CHAPTER 2

Overview

Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 9000 Series devices is as follows:

- **Resilient**

  Provides critical business-class availability.

- **Modular**

  Has extensions that accommodate business needs.

- **Highly Programmatic**

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

- **Secure**

  Protects and preserves data and operations.

- **Flexible**

  Integrates and enables new technologies.

- **Scalable**

  Accommodates and grows with the business and its requirements.

- **Easy to use**

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

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

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

Standard Network Manageability Features

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

Advanced Automation Feature

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

PowerOn Auto Provisioning Support

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

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

For more details about POAP, see the *Cisco Nexus 9000 Series NX-OS Fundamentals Configuration Guide*.

OpenStack Integration

The Cisco Nexus 9000 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 1: Summary of Cisco Nexus Plugin features for OpenStack Networking (Neutron)

Considerations	Description	Cisco Nexus Plugin
Extension of tenant VLANs across virtualization hosts	VLANs must be configured on both physical and virtual networks. OpenStack Networking supports only a single plugin at a time. You must choose which parts of the networks to manually configure.	Accepts networking API calls and configures both physical and virtual switches.
Efficient use of scarce VLAN IDs	Static provisioning of VLAN IDs on every switch rapidly consumes all available VLAN IDs, which limits scalability and makes the network vulnerable to broadcast storms.	Efficiently uses limited VLAN IDs by provisioning and deprovisioning VLANs across switches as tenant networks are created and destroyed.
Easy configuration of tenant VLANs in a top-of-rack (ToR) switch	You must statically provision all available VLANs on all physical switches. This process is manual and error prone.	Dynamically provisions tenant-network-specific VLANs on switch ports connected to virtualization hosts through the Nexus plugin driver.
Intelligent assignment of VLAN IDs	Switch ports connected to virtualization hosts are configured to handle all VLANs. Hardware limits are reached quickly.	Configures switch ports connected to virtualization hosts only for the VLANs that correspond to the networks configured on the host. This feature enables accurate port and VLAN associations.
Aggregation switch VLAN configuration for large multirack deployments.	When compute hosts run in several racks, you must fully mesh top-of-rack switches or manually trunk aggregation switches.	Supports Cisco Nexus 2000 Series Fabric Extenders to enable large, multirack deployments and eliminates the need for an aggregation switch VLAN configuration.

Programmability Support

Cisco NX-OS on Cisco Nexus 9000 Series devices support the following capabilities to aid programmability:

- NX-API support

- Python scripting
- Tcl scripting
- Broadcom Shell
- Bash
- Guest Shell

NX-API Support

Cisco NX-API allows for HTTP-based programmatic access to the Cisco Nexus 9000 Series platform. This support is delivered by NX-API, an open source webserver. NX-API provides the configuration and management capabilities of the Cisco NX-OS CLI with web-based APIs. The device can be set to publish the output of the API calls in XML or JSON format. This API enables rapid development on the Cisco Nexus 9000 Series platform.

Python Scripting

Cisco Nexus 9000 Series devices support Python v2.7.5 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.

Tcl Scripting

Cisco Nexus 9000 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.

Broadcom Shell

The Cisco Nexus 9000 Series device front panel and fabric module line cards contain Broadcom Network Forwarding Engine (NFE). You can access the Broadcom command line shell (bcm-shell) from these NFEs.

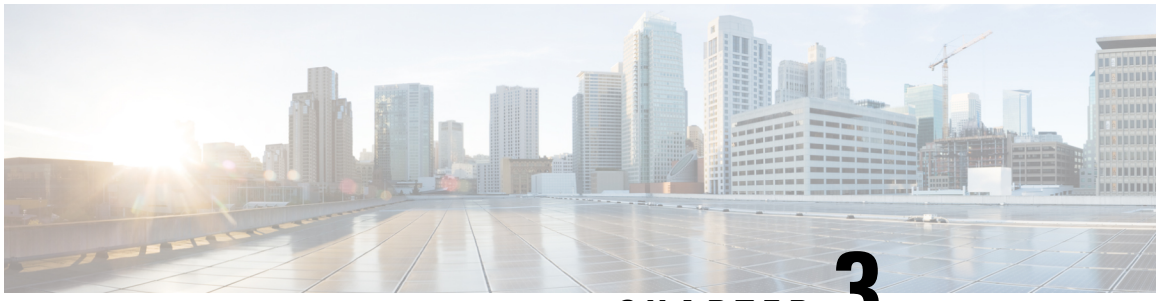
Bash

Cisco Nexus 9000 Series devices support direct Bourne-Again SHell (Bash) access. With Bash, you can access the underlying Linux system on the device and manage the system.

Guest Shell

The Cisco Nexus 9000 Series devices support a guest shell that provides Bash access into a 64-bit Linux execution space on the host system that is decoupled from the host Cisco Nexus 9000 NX-OS software. With

the guest shell you can add software packages and update libraries as needed without impacting the host system software.



CHAPTER 3

NX-API

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 9000 Series devices. NX-API supports **show** commands, configurations, and Linux Bash.

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 9000 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 9000 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.

Note For more details about NX-API response codes, see Table of NX-API Response Codes, on page 61.

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

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

- Enable the management interface.

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

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

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

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

Request:

```
<?xml version="1.0" 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/nexus9000/tree/master/nx-os/nxapi/check_cable.

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

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 9000 Series device using HTTP/HTTPS. After posting the request, the output response is displayed.

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

Use a browser to access the NX-API Sandbox.

Note When using the NX-API Sandbox, Cisco recommends that you use the Firefox browser, release 24.0 or later.

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

- Enable the management interface.

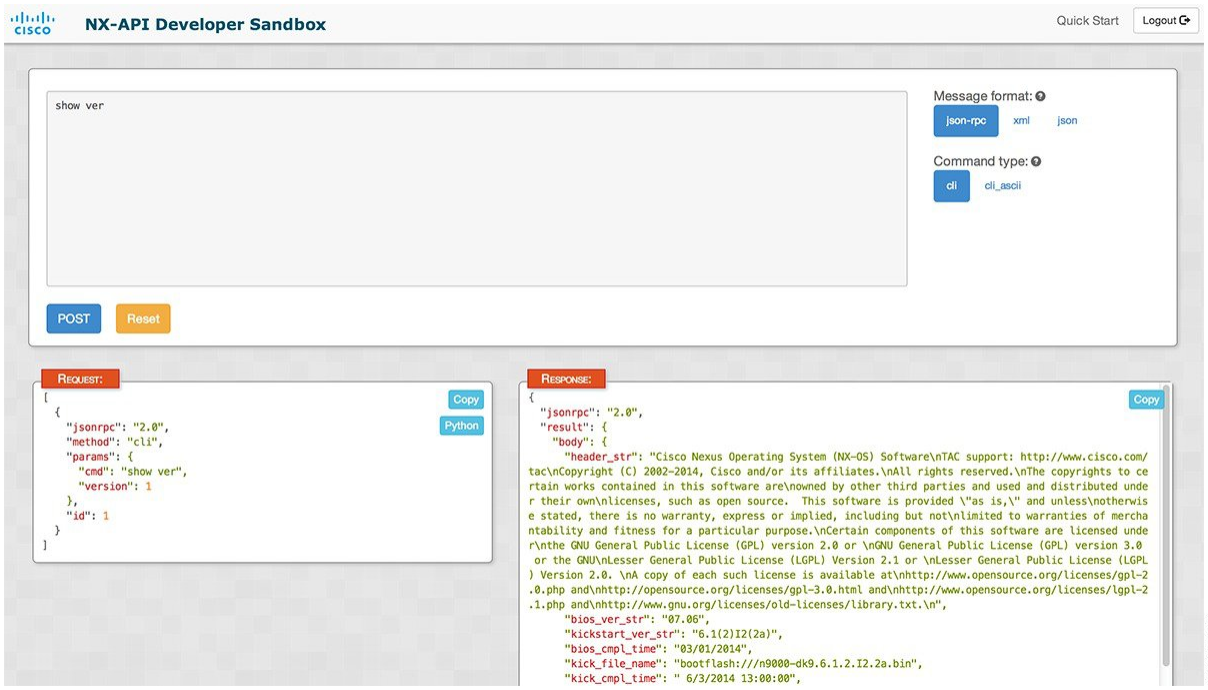
```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

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

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

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

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



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

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

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

NX-API Management Commands

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

Table 2: NX-API Management Commands

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

NX-API Management Command	Description
nxapi certificate {*httpscrt* |*httpskey*}	Specifies the upload of the following: • HTTPS certificate when *httpscrt* is specified. • HTTPS key when *httpskey* is specified.
nxapi certificate *enable*	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 3: NX-API Request Elements

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

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

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>chunk</td><td>Some show commands can return a large amount of output. For the NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for show commands.

Enable or disable chunk with the following settings:

<table><tr><td>0</td><td>Do not chunk output.</td></tr><tr><td>1</td><td>Chunk output.</td></tr></table>
Note Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.

The output message format is XML. (XML is the default.) Special characters, such as < or >, are converted to form a valid XML message (< is converted into < > is converted into &gt).

You can use XML SAX to parse the chunked output.

Note When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</td></tr>
<tr><td>sid</td><td>The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a sid to match the sid of the previous response message.</td></tr>
<tr><td>input</td><td>Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, show commands are cli_show message type and are not supported in cli_conf mode.

Note Except for bash, multiple commands are separated with " ; ". (The ; must be surrounded with single blank characters.)

For bash, multiple commands are separated with ";". (The ; is not surrounded with single blank characters.)

The following are examples of multiple commands:

<table><tr><td>cli_show</td><td><code>show version ; show interface brief ; show vlan</code></td></tr><tr><td>cli_conf</td><td><code>interface Eth4/1 ; no shut ; switchport</code></td></tr><tr><td>bash</td><td><code>cd /bootflash;mkdir new_dir</code></td></tr></table></td></tr>
</table>

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>output_format</td><td>The available output message formats are the following:
<table>
<tr><td>xml</td><td>Specifies output in XML format.</td></tr>
<tr><td>json</td><td>Specifies output in JSON format.</td></tr>
</table>
Note The Cisco Nexus 9000 Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.

To manage the computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported.

The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).</td></tr>
</table>

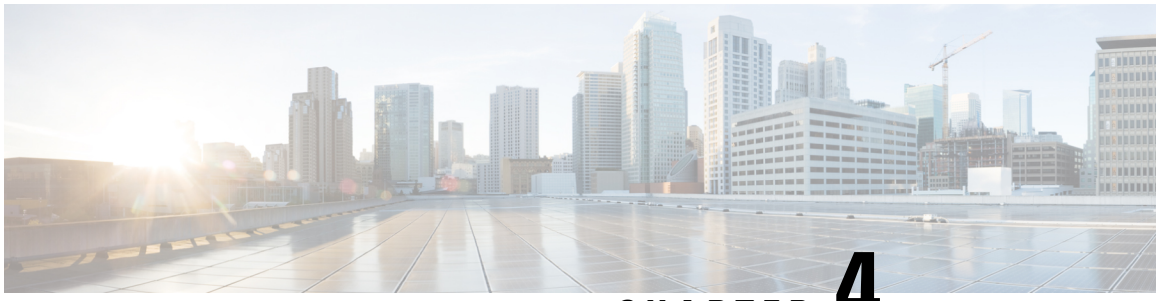
NX-API Response Elements

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

Table 4: NX-API Response Elements

NX-API Response Element	Description
version	NX-API version.
type	Type of command to be executed.
sid	Session ID of the response. This element is valid only when the response message is chunked.
outputs	Tag that encloses all command outputs. When multiple commands are in cli_show or cli_show_ascii, each command output is enclosed by a single output tag. When the message type is cli_conf or bash, there is a single output tag for all the commands because cli_conf and bash commands require context.
output	Tag that encloses the output of a single command output. For cli_conf and bash message types, this element contains the outputs of all the commands.
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.

NX-API Response Element	Description
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.



CHAPTER 4

Python API

About the Python API

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

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

http://www.python.org/

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

The Cisco Nexus 9000 Series devices support Python v2.7.5 in both interactive and non-interactive (script) modes.

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

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

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

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

you can run the help command for a specific module. For example, **help**(*cisco.interface*) displays the properties of the cisco.interface module.

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

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

NAME
    cisco

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

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

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

Using the CLI Command APIs

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

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

Table 5: CLI Command APIs

API	Description
cli() Example: `string = cli ("cli-command")`	Returns the raw output of CLI commands, including control/special characters. **Note** The interactive Python interpreter prints control/special characters 'escaped'. A carriage return is printed as '\n' and gives results that might be difficult to read. The **clip()** API gives results that are more readable.
clid() Example: `json_string = clid ("cli-command")`	Returns JSON output for cli-command, if XML support exists for the command, otherwise an exception is thrown. **Note** This API can be useful when searching the output of show commands.

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

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

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

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

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

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

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

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

Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python from the CLI:

Note The Python interpreter is designated with the ">>>" or "…" prompt.

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 5 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...   intf=intflist['TABLE_interface']['ROW_interface'][i]
...   i=i+1
...   if intf['state'] == 'up':
...     print intf['interface']
...
mgmt0
Ethernet2/7
Ethernet4/7
loopback0
loopback5
>>>
```

Display Formats

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

Example 1:

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

Example 2:

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

Example 3:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
```

```
''
>>> r = cli('where detail') ; print r
  mode:
  username:            admin
  vdc:                 EOR-1
  routing-context vrf: default
>>>
```

Example 4:

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                kern_uptm_secs = 6
                kick_file_name = bootflash:///n9000-dk9.6.1.2.I1.1.bin
                    rr_service = None
                     module_id = Supervisor Module
                   kick_tmstmp = 10/21/2013 00:06:10
                bios_cmpl_time = 08/17/2013
                bootflash_size = 20971520
             kickstart_ver_str = 6.1(2)I1(2) [build 6.1(2)I1(2)] [gdb]
                kick_cmpl_time = 10/20/2013 4:00:00
                    chassis_id = Nexus9000 C9508 (8 Slot) Chassis
                 proc_board_id = SAL171211LX
                        memory = 16077872
                  manufacturer = Cisco Systems, Inc.
                kern_uptm_mins = 26
                  bios_ver_str = 06.14
                      cpu_name = Intel(R) Xeon(R) CPU E5-2403
                 kern_uptm_hrs = 2
                       rr_usecs = 816550
                    rr_sys_ver = None
                     rr_reason = Reset Requested by CLI command reload
                      rr_ctime = Mon Oct 21 00:10:24 2013
                    header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html
Copyright (c) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained herein are owned by
other third parties and are used and distributed under license.
Some parts of this software are covered under the GNU Public
License. A copy of the license is available at
http://www.gnu.org/licenses/gpl.html.
                     host_name = switch
                      mem_type = kB
                kern_uptm_days = 0
>>>
```

Non-interactive Python

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

The Cisco Nexus 9000 Series device also supports the source CLI command for running Python scripts. The **bootflash:scripts** directory is the default script directory for the source CLI command.

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

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

from cli import *
import sys, time

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

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

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

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

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

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

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

```
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port
```

Running Scripts with Embedded Event Manager

On Cisco Nexus 9000 Series devices, embedded event manager (EEM) policies support Python scripts.

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

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

```
switch# show running-config eem

!Command: show running-config eem
!Time: Sun May  1 14:40:07 2011

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

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

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

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

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

PC_VSH_CMD_TLV(7679) with q
```

Python Integration with Cisco NX-OS Network Interfaces

On Cisco Nexus 9000 Series devices, Python is integrated with the underlying Cisco NX-OS network interfaces. You can switch from one virtual routing context to another by setting up a context through the cisco.vrf.set_global_vrf() API.

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

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

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

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

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

    Returns: Nothing

>>>
```

Cisco NX-OS Security with Python

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

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

Examples of Security and User Authority

The following example shows how a privileged user runs commands:

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

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

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

```
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

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

The following is an example for a privileged user:

```
>>> from cli import *
>>> cli('show clock')
'11:28:53.845 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

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

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

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

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

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

The following example shows an RBAC configuration:

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

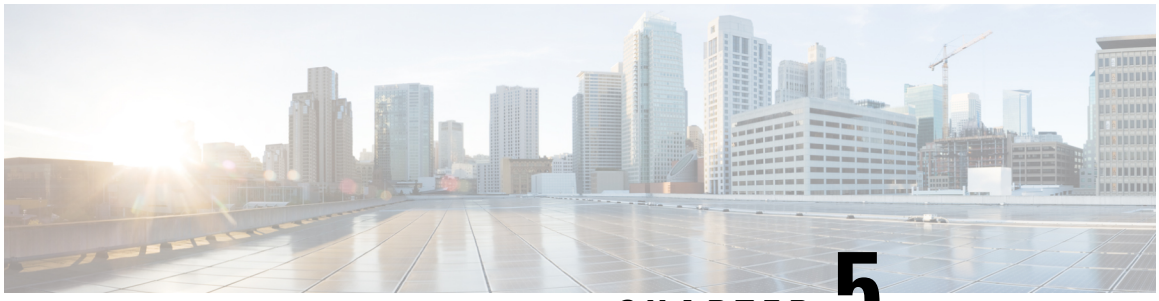
Example of Running Script with Scheduler

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

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

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

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

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



CHAPTER 5

Broadcom Shell

About the Broadcom Shell

The Cisco Nexus 9000 Series device front panel and fabric module line cards contain Broadcom Network Forwarding Engines (NFE). The number of NFEs varies depending upon the specific model of the front panel line card (LC) or the fabric module (FM).

The following sections describe how you can access the command-line shell (bcm-shell) and how to read from these NFEs.

Guidelines and Limitations

Using the Broadcom Shell has the following guideline and limitation:

- You can access and read information from the T2 ASICs without any limitations. However, Cisco does not recommend that you change the settings of the T2 configuration. Use caution when accessing the Broadcom Shell.

Accessing the Broadcom Shell (bcm-shell)

The following sections describe approaches to access the Broadcom Shell (bcm-shell).

Accessing bcm-shell with the CLI API

The bcm-shell commands are passed directly from the Cisco Nexus 9000 Series CLI to the specific T2 ASIC instance. The T2 ASIC instance can be on the fabric module or on the front panel line card.

The command syntax is as follows:

bcm-shell module *module_number* [*instance_number:command*]

where

module_number	Module number in the chassis.

instance_number	T2 instance number • When not specified, the T2 instance number defaults to 0. • When a wildcard ('*') is specified, all T2 instances are processed.
command	Broadcom command

Note Cisco NX-OS command extensions such as ‘pipe include’ or ‘redirect output to file’ can be used to manage command output.

Note Entering commands with the CLI API are recorded in the system accounting log for auditing purposes. Commands entered directly from the bcm-shell are not recorded in the accounting log.

Accessing the Native bcm-shell on the Fabric Module

An eight-slot line card (LC) chassis can host a maximum of six fabric modules (FMs). These slots are numbered 21 through 26.You must specify the FM that you wish to access the bcm-shell on.

The following example shows how to access the bcm-shell on the FM in slot 24, access context help, and exit the bcm-shell.

- Use the show module command to display the FMs.

```
n9k-spine1# show module
Mod Ports Module-Type Model Status
--- ----- -------------------------------- ------------------ ----------
3 36 36p 40G Ethernet Module N9k-X9636PQ ok
4 36 36p 40G Ethernet Module N9k-X9636PQ ok
21 0 Fabric Module N9K-C9508-FM ok
22 0 Fabric Module N9K-C9508-FM ok
23 0 Fabric Module N9K-C9508-FM ok
24 0 Fabric Module N9K-C9508-FM ok
25 0 Fabric Module N9K-C9508-FM ok
26 0 Fabric Module N9K-C9508-FM ok
27 0 Supervisor Module N9K-SUP-A active *
29 0 System Controller N9K-SC-A active
```

- Attach to module 24 to gain access to the command line for the FM in slot 24.

```
n9k-spine1# attach module 24
Attaching to module 24 ...
To exit type 'exit', to abort type '$.'
```

- Enter the command to gain root access to the fabric module software.

```
module-24# test hardware internal bcm-usd bcm-diag-shell
Available Unit Numbers: 0 1
bcm-shell.0> 1
```

  At this point, you are at the Broadcom shell for the fabric module in slot 24, T2 ASIC instance 1. Any commands entered are specific to this specific ASIC instance.

- Use the exit command to exit the bcm-shell and to detach from the FM.

```
bcm-shell.1> exit
module-24# exit
rlogin: connection closed.
```

Accessing the bcm-shell on the Line Card

When connecting to the T2 ASIC on the line card (LC), you first attach to the module, enter root mode, run the shell access exec, and select the ASIC instance that you want to attach to. The number of available ASICs depends on the model of the line card you are attached to.

The following example shows how to access the bcm-shell of ASIC instance 1 on the LC in slot 2 and exit the bcm-shell on a LC that contains three T2 instances.

- Attach to module 2 to gain access to the command line for the LC in slot 2.

```
n9k-spine1# attach module 2
Attaching to module 2 ...
To exit type 'exit', to abort type '$.'
Last login: Wed Aug 7 14:13:15 UTC 2013 from sup27 on ttyp0
```

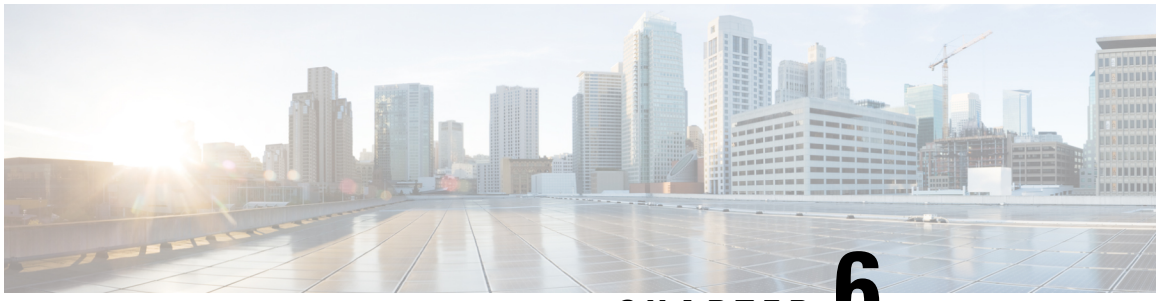
- Enter the command to gain root access to the line card software.

```
module-2# test hardware internal bcm-usd bcm-diag-shell
Available Unit Numbers: 0 1 2
bcm-shell.0> 1
bcm-shell.1>
```

  At this point you are at the Broadcom shell for the line card module in slot 2, T2 ASIC instance 1.

- Use the exit command to exit the bcm-shell and detach from the FM.

```
bcm-shell.1> exit
module-2# exit
rlogin: connection closed.
```



CHAPTER 6

Bash

About Bash

In addition to the NX-OS CLI, Cisco Nexus 9000 Series devices support access to the Bourne-Again SHell (Bash). Bash interprets commands that you enter or commands that are read from a shell script. Using Bash enables access to the underlying Linux system on the device and to manage the system.

Accessing Bash

In Cisco NX-OS, Bash is accessible from user accounts that are associated with the Cisco NX-OS dev-ops role or the Cisco NX-OS network-admin role.

The following example shows the authority of the dev-ops role and the network-admin role:

```
switch# show role name dev-ops

Role: dev-ops
  Description: Predefined system role for devops access. This role
  cannot be modified.
  Vlan policy: permit (default)
  Interface policy: permit (default)
  Vrf policy: permit (default)
  ----------------------------------------------------------------------
  Rule    Perm    Type        Scope               Entity
  ----------------------------------------------------------------------
  4       permit  command                         conf t ; username *
  3       permit  command                         bcm module *
  2       permit  command                         run bash *
  1       permit  command                         python *

switch# show role name network-admin

Role: network-admin
  Description: Predefined network admin role has access to all commands
  on the switch
  ----------------------------------------------------------------------
```

```
  Rule    Perm    Type        Scope               Entity
  ----------------------------------------------------------------------
  1       permit  read-write
switch#
```

Bash is enabled by running the **feature bash-shell** command.

The **run bash** command loads Bash and begins at the home directory for the user.

The following examples show how to enable the Bash shell feature and how to run Bash.

```
switch# config t
switch(config)# feature bash-shell
```

```
switch# run?
  run         Execute/run program
  run-script  Run shell scripts

switch# run bash?
  bash  Linux-bash

switch# run bash
bash-4.2$ whoami
admin
bash-4.2$ pwd
/bootflash/home/admin
bash-4.2$
```

Note You can also execute Bash commands with the **run bash** *command* command.

The following is an example of the **run bash** *command* command.

```
run bash whoami
```

Escalate Privileges to Root

The privileges of an admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- admin privilege user (network-admin / vdc-admin) is equivalent of Linux root privilege user in NX-OS
- Only an authenticated admin user can escalate privileges to root, and password is not required for and authenticated admin privilege user.
- Bash must be enabled before escalating privileges.

The following example shows how to escalate privileges to root and how to verify the escalation:

```
switch# run bash
bash-4.2$ sudo su root

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
```

```
    #3) With great power comes great responsibility.

Password:

bash-4.2# whoami
root
bash-4.2# exit
exit
```

Examples of Bash Commands

This section contains examples of Bash commands and output.

Displaying System Statistics

The following example shows how to display system statistics:

```
switch# run bash
bash-4.2$ cat /proc/meminfo
<snip>
MemTotal:       16402560 kB
MemFree:        14098136 kB
Buffers:           11492 kB
Cached:          1287880 kB
SwapCached:            0 kB
Active:          1109448 kB
Inactive:         717036 kB
Active(anon):     817856 kB
Inactive(anon):   702880 kB
Active(file):     291592 kB
Inactive(file):    14156 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                32 kB
Writeback:             0 kB
AnonPages:        527088 kB
Mapped:            97832 kB
<\snip>
```

Running Bash from CLI

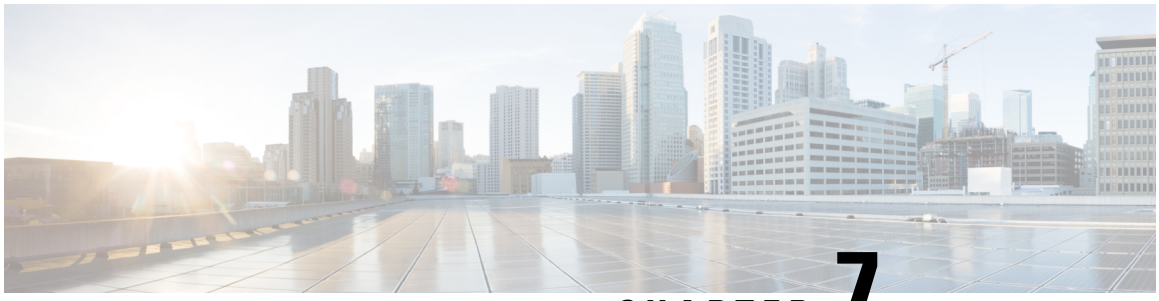
The following example shows how to run a bash command from the CLI with the **run bash** *command* command:

```
switch# run bash ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   528 poll_s ?        00:00:03 init
1 S     0     2     0  0  80   0 -     0 kthrea ?        00:00:00 kthreadd
1 S     0     3     2  0  80   0 -     0 run_ks ?        00:00:56 ksoftirqd/0
1 S     0     6     2  0 -40   - -     0 cpu_st ?        00:00:00 migration/0
1 S     0     7     2  0 -40   - -     0 watchd ?        00:00:00 watchdog/0
1 S     0     8     2  0 -40   - -     0 cpu_st ?        00:00:00 migration/1
1 S     0     9     2  0  80   0 -     0 worker ?        00:00:00 kworker/1:0
1 S     0    10     2  0  80   0 -     0 run_ks ?        00:00:00 ksoftirqd/1
```

Running Python from Bash

The following example shows how to load Python and configure a switch using Python objects:

```
switch# run bash
bash-4.2$ python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

!Command: show running-config interface Ethernet1/3
!Time: Mon Nov 4 13:17:56 2013

version 6.1(2)I2(1)

interface Ethernet1/3
  vrf member myvrf

>>>
```



CHAPTER 7

Guest Shell

About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, the Cisco Nexus 9000 Series devices support access to a decoupled execution space running within a Linux Container (LXC) called the “guest shell”.

From within the guest shell the network-admin has the following capabilities:

- Access to the network.
- Access to Cisco Nexus 9000 bootflash.
- Access to Cisco Nexus 9000 CLI.
- Access to Cisco onePK APIs.
- The ability to install and run python scripts.
- The ability to install and run 64-bit Linux applications.

Decoupling the execution space from the native host system allows customization of the Linux environment to suit the needs of the applications without impacting the host system or applications running in other Linux Containers.

On NX-OS devices, Linux Containers are installed and managed with the virtual-service commands. The guest shell will appear in the virtual-service show command output.

Note By default the guest shell occupies approximately 5 MB of RAM and 200 MB of bootflash when enabled. Use the **guestshell destroy** command to reclaim resources if the guest shell is not used.

Accessing the Guest Shell

In Cisco NX-OS, the guest shell is accessible to the network-admin. It is automatically enabled in the system and can be accessed using the **run guestshell** command. Consistent with the **run bash** command, these commands can be issued within the guest shell with the **run guestshell** *command* form of the NX-OS CLI command.

```
switch# run guestshell ls -al /bootflash/*.ova
-rw-rw-rw- 1 2002 503 117616640 Aug 21 18:04 /bootflash/chef.ova
-rw-rw-rw- 1 2002 503 83814400 Aug 21 18:04 /bootflash/pup.ova
-rw-rw-rw- 1 2002 503 40724480 Apr 15 2012 /bootflash/red.ova
```

Note When running in the guest shell, you have network-admin level privileges.

Capabilities in the Guest Shell

The guest shell has a number of utilities and capabilities available by default.

NX-OS CLI in the Guest Shell

The guest shell provides an application to allow the user to issue NX-OS commands from the guest shell environment to the host network element. The **dohost** application accepts any valid NX-OS configuration or exec commands and issues them to the host network element.

When invoking the **dohost** command each NX-OS command must be in double quotes:

```
dohost "<NXOS CLI>"
```

The NX-OS CLI can be chained together:

```
guestshell:~$ dohost "conf t ; cdp timer 20"
{0}{}
{0}{}
guestshell:~$ dohost "show run | inc cdp"
{0}{cdp timer 20}
```

The value between the first set of brackets is the result code from the NXOS parser. The value between the second set of brackets is the output result from the command issued.

The NX-OS CLI can also be chained together using the NX-OS style command chaining technique by adding a semicolon between each command. (A space on either side of the semicolon is required.):

```
guestshell:~$ dohost "conf t ; cdp timer 13 ; show run | inc cdp"
{0}{cdp timer 13}
```

In this example, the **dohost** command received only one quoted command string. It returns one result string back from the NX-OS parser.

Note Commands issued on the host through the **dohost** command are run with network-admin level privileges.

Network Access in Guest Shell

The guest shell has a number of typical network utilities included by default and they can be used on different VRFs using the **chvrf** *vrf command* command.

Note Commands that are run without the **chvrf** command are run within the context of the default VRF.

For example, to ping IP address 10.0.0.1 over the management VRF, the command is "**chvrf** *management ping 10.0.0.1*". Other utilities such as **scp** or **ssh** would be similar.

Example:

```
switch# guestshell
guestshell:~$ cd /bootflash
guestshell:/bootflash$ chvrf management scp foo@10.28.38.48:/foo/index.html index.html
foo@10.28.38.48's password:
index.html 100% 1804 1.8KB/s 00:00
guestshell:/bootflash$ ls -al index.html
-rw-r--r-- 1 guestshe users 1804 Sep 13 20:28 index.html
guestshell:/bootflash$
guestshell:/bootflash$ chvrf management curl cisco.com
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.cisco.com/">here</a>.</p>
</body></html>
guestshell:/bootflash$
```

To obtain a list of VRFs on the system, use the **show vrf** command. The command can be run natively from the NX-OS CLI or by with the dohost *show vrf* command in the guest shell.

Example:

```
guestshell:/bootflash$ dohost "show vrf"
{0}{VRF-Name VRF-ID State Reason
default 1 Up --
management 2 Up --}
```

To resolve domain names from within the guest shell, the resolver needs to be configured. Edit the /etc/resolv.conf file in the guest shell to include a DNS nameserver and domain as appropriate for the network.

Example:

```
nameserver 10.1.1.1
domain cisco.com
```

The nameserver and domain information should match what is configured through the NX-OS configuration.

Example:

```
switch(config)# ip domain-name cisco.com
switch(config)# ip name-server 10.1.1.1
switch(config)# vrf context management
switch(config-vrf)# ip domain-name cisco.com
switch(config-vrf)# ip name-server 10.1.1.1
```

If the Cisco Nexus 9000 device is in a network that uses an HTTP proxy server, the **http_proxy** and **https_proxy** environment variables must be set up within the guest shell also.

Example:

```
export http_proxy=http://proxy.esl.cisco.com:8080
export https_proxy=http://proxy.esl.cisco.com:8080
```

These environment variables should be set in the .bashrc file or in an appropriate script to ensure that they are persistent.

Access to Bootflash in Guest Shell

Network administrators can manage files with Linux commands and utilities in addition to using NX-OS CLI commands. By mounting the system bootflash at /bootflash in the guest shell environment, the network-admin can operate on these files with Linux commands.

Example:

```
find . –name “foo.txt”
rm “/bootflash/junk/foo.txt”
```

Python in Guest Shell

Python can be used interactively or python scripts can be run in the guest shell.

Example:

```
guestshell:~$ python
Python 2.7.3 (default, Aug 22 2014, 12:09:58)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
guestshell:~$
```

The pip python package manager is included in the guest shell to allow the network-admin to install new python packages.

Example:

```
guestshell:~$ pip list
argparse (1.2.1)
async (0.6.1)
iniparse (0.3.2)
ipaddress (branches-3144)
pexpect (2.3)
pip (1.5.6)
pycurl (7.19.0)
setuptools (0.6c11)
smart (1.4.1)
urlgrabber (3.9.1)
yum-metadata-parser (1.1.4)
```

Installing RPMs in the Guest Shell

By default, the Yum RPM package manager is included in the guest shell for the installation of software packages. Yum is pointed to the yocto repository.

```
guestshell:~$ cat /etc/yum/repos.d/yumrepo_x86_64.repo
[poky_1_5_1_x86_64]
baseurl=http://downloads.yoctoproject.org/releases/yocto/yocto-1.5.1/rpm/x86_64/
name=Poky 1.5.1 repository (x86_64)
enabled=1
```

Yum can be pointed to one or more repositories at any time by modifying the `yumrepo_x86_64.repo` file or by adding a new `.repo` file in the `repos.d` directory.

Yum resolves the dependancies and installs all the required packages.

```
guestshell:~$ sudo chvrf management yum install perl
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package perl.x86_64 0:5.14.3-r1 set to be updated
--> Processing Dependency: libperl5 >= 5.14.3 for package: perl-5.14.3-r1.x86_64
--> Processing Dependency: libperl.so.5()(64bit) for package: perl-5.14.3-r1.x86_64
--> Running transaction check
---> Package libperl5.x86_64 0:5.14.3-r1 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch          Version           Repository                Size
================================================================================
Installing:
 perl            x86_64        5.14.3-r1         poky_1_5_1_x86_64         16 k
Installing for dependencies:
 libperl5        x86_64        5.14.3-r1         poky_1_5_1_x86_64        712 k

Transaction Summary
================================================================================
Install       2 Package(s)
Upgrade       0 Package(s)

Total size: 728 k
Installed size: 1.6 M
```

```
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : libperl5-5.14.3-r1.x86_64                                  1/2
  Installing     : perl-5.14.3-r1.x86_64                                      2/2

Installed:
  perl.x86_64 0:5.14.3-r1

Dependency Installed:
  libperl5.x86_64 0:5.14.3-r1

Complete!
guestshell:~$
```

Note When more space is needed in the guest shell root file system for installing or running packages, the **guestshell resize roofs** *size-in-MB* command is used to increase the size of the file system.

Note Some open source software packages from the repository might not install or run as expected in the guest shell as a result of restrictions that have been put into place to protect the integrity of the host system.

Resources Used for the Guest Shell

By default, the resources for the guest shell have a small impact on resources available for normal switch operations. If the network-admin requires additional resources for the guest shell, the **guestshell resize** {*cpu* | *memory* | *rootfs*} command changes these limits.

Resource	Default	Minimum/Maximum
CPU	1%	1/20%
Memory	256MB	256/3840MB
Storage	204MB	204/1024MB

The CPU limit is the percentage of the system compute capacity that tasks running within the guest shell are given when there is contention with other compute loads in the system. When there is no contention for CPU resources, the tasks within the guest shell are not limited.

Note A guest shell reboot is required after changing the resource allocations. This can be accomplished with the **guestshell reboot** command.

Security Posture for Virtual Services

Use of the guest shell and virtual services in Cisco Nexus 9000 series devices are only two of the many ways that the network-admin can manage or extend the functionality of the system. These options are geared towards providing an execution environment that is decoupled from the native host context. This separation allows the introduction of software into the system that may not be compatible with the native execution environment. It also allows the software to run in an environment that does not interfere with the behavior, performance, or scale of the system.

Digitally Signed Application Packages

By default, Cisco network elements require applications to provide a valid Cisco digital signature at runtime. The Cisco digital signature ensures the integrity of Cisco-developed packages and applications.

The Cisco Nexus 9000 Series switches support the configuration of a signing level policy to allow for unsigned OVA software packages. To allow unsigned and Cisco-signed packages for creating virtual-services, the network-admin can configure the following:

```
virtual-service
    signing level unsigned
```

Note The guest shell software package has a Cisco signature and does not require this configuration.

Kernel Vulnerability Patches

Cisco responds to pertinent Common Vulnerabilities and Exposures (CVEs) with platform updates that address known vulnerabilities.

ASLR and X-Space Support

Cisco Nexus 9000 NX-OS supports the use of Address Space Layout Randomization (ASLR) and Executable Space Protection (X-Space) for runtime defense. The software in Cisco-signed packages make use of this capability. If other software is installed on the system, it is recommended that it be built using a host OS and development toolchain that supports these technologies. Doing so reduces the potential attack surface that the software presents to potential intruders.

Root-User Restrictions

As a best practice for developing secure code, it is recommend running applications with the least privilege needed to accomplish the assigned task. To help prevent unintended accesses, software added into the guest shell should follow this best practice.

All processes within a virtual service are subject to restrictions imposed by reduced Linux capabilities. If your application must perform operations that require root privileges, restrict the use of the root account to the smallest set of operations that absolutely requires root access, and impose other controls such as a hard limit on the amount of time that the application can run in that mode.

The set of Linux capabilities that are dropped for root within virtual services follow:

CAP_SYS_BOOT	CAP_MKNOD	CAP_SYS_PACCT
CAP_SYS_MODULE	CAP_MAC_OVERRIDE	CAP_SYS_RESOURCE
CAP_SYS_TIME	CAP_SYS_RAWIO	CAP_AUDIT_WRITE
CAP_AUDIT_CONTROL	CAP_SYS_NICE	CAP_SETFCAP
CAP_MAC_ADMIN	CAP_SYS_PTRACE	CAP_SETPCAP

As root within a virtual-service, bind mounts may be used as well as tmpfs and ramfs mounts. Other mounts are prevented.

Namespace Isolation

The host and virtual service are separated into separate namespaces. This provides the basis of separating the execution spaces of the virtual services from the host. Namespace isolation helps to protect against data loss and data corruption due to accidental or intentional data overwrites between trust boundaries. It also helps to ensure the integrity of confidential data by preventing data leakage between trust boundaries: an application in one virtual service cannot access data in another virtual service

Guest File System Access Restrictions

To preserve the integrity of the files within the virtual services, the file systems of the virtual services are not accessible from the NX-OS CLI. If a given virtual-service allows files to be modified, it needs to provide an alternate means by which this can be done (i.e. yum install, scp, ftp, etc).

The guest shell mounts the `bootflash` of the host system at `/bootflash`. The network-admin can access the file using an NX-OS CLI or Linux command from within the guest shell.

Resource Management

A Denial-of-Service (DoS) attack attempts to make a machine or network resource unavailable to its intended users. Misbehaving or malicious application code can cause DoS as the result of over-consumption of connection bandwidth, disk space, memory, and other resources. The host provides resource-management features that ensure fair allocation of resources among all virtual services on the host.

Secure IPC

Applications in a guest shell or virtual service can be made more integrated with the host by using Cisco onePK services. The applications communicate with the host network element over TIPC. Applications within various containers are not allowed to communicate with each other over TIPC, they are only allowed to talk to the host. This prevents issues of one container from spoofing that it is where the Cisco onePK services are running. Applications in containers are also not allowed to listen on TIPC ports.

To ensure that only know virtual services can communicate with the host, a unique identifier for each virtual service is created when it is enabled and verified at the time when the onePK communication channel is established.

The system also limits the rate at which an application in an individual virtual service can send messages to the host. This behavior prevents a misbehaving application from sending messages frequently enough to prevent normal operation of the host or to block other virtual services on the same host from communicating with the host.

Guidelines and Limitations

The guest shell has the following guidelines and limitations:

- By default, the guest shell starts an Open-SSH v6.2p2 server upon boot up. The server listens on port 4022 on the localhost ip address interface 127.0.0.1 only. This provides the password-less connectivity into the guest shell from the NX-OS vegas-shell when the **guestshell** keyword is entered. If this server is killed or its configuration (residing in `/etc/ssh/sshd_config`) is altered, access to the guest shell from the NX-OS CLI may not work. When this happens, you should navigate back into the guest shell with the **virtual-service connect name guestshell+ console** command. The username/password for this access is `guestshell/guestshell`.

  To instantiate your own Open-SSH server within the guest shell use the following steps as root:

  - Determine which VRF you want to establish your ssh connections through.
  - Determine which port you want your Open-SSH server to listen for connections. Use the NX-OS CLI **show socket connection** command to view which ports that are already in use.

    Note Do not select port 4022.

  - Start your Open-SSH server with the following command:

    chvrf *vrf_name /usr/sbin/sshd -p port_number*

- The time zone within the guest shell is not updated when the **clock timezone** command is configured.

  Use the Linux *TZ* environment variable to change the timezone within the guest shell.

  The following is an example that configures the time zone in the guest shell:

```
switch(config)# clock timezone PDT -7 0
switch(config)# clock set 11:01:15 12 Sep 2014
Fri Sep 12 11:01:15 PDT 2014
switch(config)# show clock
11:01:26.421 PDT Fri Sep 12 2014
switch(config)# run guestshell
guestshell:~$ export TZ='PDT7'
guestshell:~$ date
Fri Sep 12 11:01:59 PDT 2014
```

- Cisco Nexus 9000 NX-OS automatically installs and enables the guest shell by default. However, if the device is reloaded with a Cisco NX-OS image that does not provide guest shell support, the existing guest shell is automatically removed and a **%VMAN-2-INVALID_PACKAGE** message is issued.

  As a best practice, remove the guest shell with the **guestshell destroy** command before reloading the older Cisco NX-OS image.

Use the **install all** command to validate the compatibility between the current Cisco NX-OS image and the target Cisco NX-OS image.

The following is an example of incompatible images:

```
switch# install all nxos n9kpregs.bin
Installer will perform compatibility check first. Please wait.
uri is: /n9kpregs.bin
2014 Aug 29 20:08:51 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE:
Successfully activated virtual service 'guestshell+'
Verifying image bootflash:/n9kpregs.bin for boot variable "nxos".
[####################] 100% -- SUCCESS
Verifying image type.
[####################] 100% -- SUCCESS
Preparing "lcn9k" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "bios" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "lcn9k" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "lcn9k" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "nxos" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "lcn9k" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
Preparing "lcn9k" version info using image bootflash:/n9kpregs.bin.
[####################] 100% -- SUCCESS
"Running-config contains configuration that is incompatible with the new image (strict
 incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out which feature
needs to be disabled.".
Performing module support checks.
[####################] 100% -- SUCCESS
Notifying services about system upgrade.
[#                    ]   0% -- FAIL.
Return code 0x42DD0006 ((null)).
"Running-config contains configuration that is incompatible with the new image (strict
 incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out
which feature needs to be disabled."
Service "vman" in vdc 1: Guest shell not supported, do 'guestshell destroy' to remove
it and then retry ISSU
Pre-upgrade check failed. Return code 0x42DD0006 ((null)).
switch#
```

Managing the Guest Shell

The following are commands to manage the guest shell:

Table 6: Guest Shell CLI Commands

Commands	Description
guestshell enable [**package** *guest shell OVA file*]	Installs and activates the guest shell using the OVA that is embedded in the system image. Installs and activates the guest shell using the specified software package (OVA file) or the embedded package from the system image (when no package is specified). Initially, guest shell packages are only available by being embedded in the system image. When the guest shell is already installed, this command enables the installed guest shell. Typically this is used after a **guestshell disable** command.
guestshell disable	Shuts down and disables the guest shell.
guestshell upgrade [**package** *guest shell OVA file*]	Deactivates and upgrades the guest shell using the specified software package (OVA file) or the embedded package from the system image (if no package is specified). Initially guest shell packages are only available by being embedded in the system image. The current rootfs for the guest shell is replaced with the rootfs in the software package. The guest shell does not make use of secondary filesystems that persist across an upgrade. Without persistent secondary filesystems, a **guestshell destroy** command followed by a **guestshell enable** command could also be used to replace the rootfs. When an upgrade is successful, the guest shell is activated. You are prompted for a confirmation prior to carrying out the upgrade command.

Commands	Description
guestshell reboot	Deactivates the guest shell and then reactivates it. You are prompted for a confirmation prior to carrying out the reboot command. **Note** This is the equivalent of a **guestshell disable** command followed by a **guestshell enable** command in exec mode. This is useful when processes inside the guest shell have been stopped and need to be restarted. The **run guestshell** command relies on `sshd` running in the guest shell. If the command does not work, the `sshd` process may have been inadvertently stopped. Performing a reboot of the guest shell from the NX-OS CLI allows it to restart and restore the command.
guestshell destroy	Deactivates and uninstalls the guest shell. All resources associated with the guest shell are returned to the system. The **show virtual-service global** command indicates when these resources become available. Issuing this command results in a prompt for a confirmation prior to carrying out the destroy command.
guestshell **run guestshell**	Connects to the guest shell that is already running with a shell prompt. No username/password is required.
guestshell run *command* **run guestshell** *command*	Executes a Linux/UNIX command within the context of the guest shell environment. After execution of the command you are returned to the switch prompt.
guestshell resize [**cpu** | **memory** | **rootfs**]	Changes the allotted resources available for the guest shell. The changes take effect the next time the guest shell is enabled or rebooted.

Commands	Description
guestshell sync	On systems that have active and standby supervisors, this command synchronizes the guest shell contents from the active supervisor to the standby supervisor. The network-admin issues this command when the guest shell rootfs has been set up to a point that they would want the same rootfs used on the standby supervisor when it becomes the active supervisor. If this command is not used, the guest shell is freshly installed when the standby supervisor transitions to an active role using the guest shell package available on that supervisor.
virtual-service reset force	In the event that the guestshell or virtual-services cannot be managed, even after a system reload, the reset command is used to force the removal of the guest shell and all virtual-services. The system needs to be reloaded for the cleanup to happen. No guest shell or additional virtual-services can be installed or enabled after issuing this command until after the system has been reloaded. You are prompted for a confirmation prior to initiating the reset.

Note Administrative privileges are necessary to enable/disable and to gain access to the guest shell environment.

Note The guest shell is implemented as a Linux container (LXC) on the host system. On NX-OS devices, LXCs are installed and managed with the virtual-service commands. The guest shell appears in the virtual-service commands as a virtual service named `guestshell+`.

Disabling the Guest Shell

The **guestshell disable** command shuts down and disables the guest shell.

When the guest shell is disabled and the system is reloaded, the guest shell remains disabled.

Example:

```
switch# show virtual-service list
Virtual Service List:
Name                   Status          Package Name
------------------------------------------------------------
guestshell+            Activated        guestshe11.ova
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n) y
```

```
2014 Jul 30 19:47:23 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2014 Jul 30 18:47:29 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
 virtual service 'guestshell+'
switch# show virtual-service list
Virtual Service List:
Name                 Status              Package Name
guestshell+          Deactivated         guestshell.ova
```

Note The guest shell is reactivated with the **guestshell enable** command.

Destroying the Guest Shell

The **guestshell destroy** command uninstalls the guest shell and its artifacts. The command does not remove the guest shell OVA.

When the guest shell is destroyed and the system is reloaded, the guest shell remains destroyed.

```
switch# show virtual-service list
Virtual Service List:
Name               Status          Package Name
------------------------------------------------
guestshell+        Deactivated     guestshell.ova

switch# guestshell destroy

You are about to destroy the guest shell and all of its contents. Be sure to save your work.
 Are you sure you want to continue? (y/n) [n] y
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Destroying virtual service
 'guestshell+'
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Successfully destroyed
virtual service 'guestshell +'

switch# show virtual-service list
Virtual Service List:
```

Note The guest shell can be re-enabled with the **guestshell enable** command.

Note In the Cisco NX-OS software, the **oneP** feature is automatically enabled for local access when a container is installed. Since the guest shell is a container, the **oneP** feature is automatically started.

If you do not want to use the guest shell, you can remove it with the **guestshell destroy** command. Once the guest shell has been removed, it remains removed for subsequent reloads. This means that when the guest shell container has been removed and the switch is reloaded, the guest shell container and the **oneP** feature are not automatically started.

Enabling the Guest Shell

The **guestshell enable** command installs the guest shell from a guest shell software package. By default, the package embedded in the system image is used for the installation. The command is also used to reactivate the guest shell if it has been disabled.

When the guest shell is enabled and the system is reloaded, the guest shell remains enabled.

Example:

```
switch# show virtual-service list
Virtual Service List:
switch# guestshell enable
2014 Jul 30 18:50:27 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual service
'guestshell+'
2014 Jul 30 18;50;42 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating

2014 Jul 30 18:50:42 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
 'guestshell+'
2014 Jul 30 18:51:16 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'

switch# show virtual-service list
Virtual Service List:
Name                   Status             Package Name
guestshell+            Activated          guestshell.ova
```

Verifying Virtual Service and Guest Shell Information

You can verify virtual service and guest shell information with the following commands:

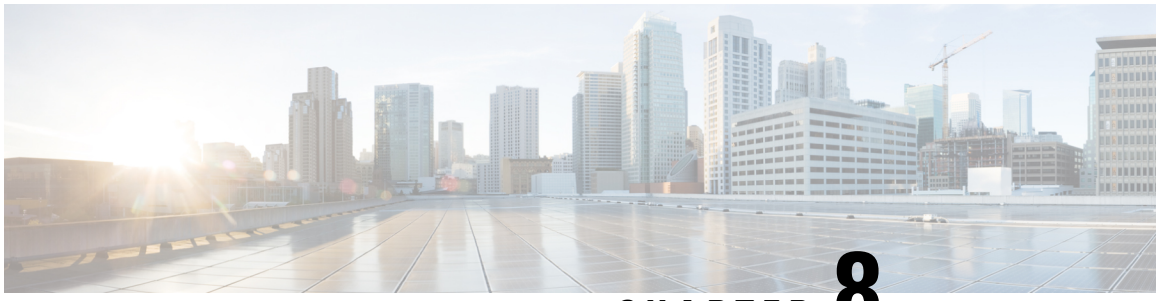
Command	Description

show virtual-service global
 	
`switch# show virtual-service global`
 	
`Virtual Service Global State and Virtualization Limits:`
 	
`Infrastructure version : 1.8`
`Total virtual services installed : 1`
`Total virtual services activated : 1`
 	
`Machine types supported : LXC`
`Machine types disabled : KVM`
 	
`Maximum VCPUs per virtual service : 1`
 	
`Resource virtualization limits:`
 	
`Name Quota Committed Available`
`---`
`system CPU (%) 6 1 5`
`memory (MB) 2304 256 2048`
`bootflash (MB) 8192 248`
`3710`
	Displays the global state and limits for virtual services.

show virtual-service list
 	
`switch# show virtual-service list *`
 	
`Virtual Service List:`
 	
`Name Status Package Name`
`---`
`guestshell+ Activated guestshell.ova`
`chef Installed chef-0.8.1-n9000-spa-k9.ova`
	Displays a summary of the virtual services, the status of the virtual services, and installed software packages.

Command	Description

show guestshell detail
 	
`switch# show guestshell detail`
`Virtual service guestshell+ detail`
`  State                 : Activated`
`  Package information`
`    Name                : guestshell.ova`
`    Path                : /isan/bin/guestshell.ova`
`    Application`
`      Name              : GuestShell`
`      Installed version : 1.0(0.1)`
`      Description       : Cisco Systems Guest Shell`
`    Signing`
`      Key type          : Cisco key`
`      Method            : SHA-1`
`    Licensing`
`      Name              : None`
`      Version           : None`
`  Resource reservation`
`    Disk                : 204 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`
	Displays details about the guestshell package (such as version, signing resources, and devices).



CHAPTER 8

Scripting with Tcl

About Tcl

Tcl (Tool Command Language) is a scripting language. With tcl, you gain more flexibility in your use of the CLI commands on the device. 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 Embedded Event Manager (EEM) policies in a script.

This section describes how to run tcl scripts or run tcl interactively on Cisco NX-OS devices.

Tclsh Command Help

Command help is not available for tcl commands. You can still access the help functions of Cisco NX-OS commands from within an interactive tcl shell.

This example shows the lack of tcl command help in an interactive tcl shell:

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# puts ?
             ^
% Invalid command at '^' marker.
switch-tcl# configure ?
  <CR>
  session   Configure the system in a session
  terminal  Configure the system from terminal input

switch-tcl#
```

Note In the above example, the Cisco NX-OS command help function is still available but the tcl **puts** command returns an error from the help function.

Tclsh Command History

You can use the arrow keys on your terminal to access commands you previously entered in the interactive tcl shell.

Note The **tclsh** command history is not saved when you exit the interactive tcl shell.

Tclsh Tab Completion

You can use tab completion for Cisco NX-OS commands when you are running an interactive tcl shell. Tab completion is not available for tcl commands.

Tclsh CLI Command

Although you can directly access Cisco NX-OS commands from within an interactive tcl shell, you can only execute Cisco NX-OS commands in a tcl script if they are prepended with the tcl **cli** command.

In an interactive tcl shell, the following commands are identical and will execute properly:

```
switch-tcl# cli show module 1 | incl Mod
switch-tcl# cli "show module 1 | incl Mod"
switch-tcl# show module 1 | incl Mod
```

In a tcl script, you must prepend Cisco NX-OS commands with the tcl **cli** command as shown in the following example:

```
set x 1
cli show module $x | incl Mod
cli "show module $x | incl Mod"
```

If you use the following commands in your script, the script will fail and the tcl shell will display an error:

```
show module $x | incl Mod
"show module $x | incl Mod"
```

Tclsh Command Separation

The semicolon (**;**) is the command separator in both Cisco NX-OS and tcl. To execute multiple Cisco NX-OS commands in a tcl command, you must enclose the Cisco NX-OS commands in quotes (**""**).

In an interactive tcl shell, the following commands are identical and will execute properly:

```
switch-tcl# cli "configure terminal ; interface loopback 10 ; description loop10"
switch-tcl# cli configure terminal ; cli interface loopback 10 ; cli description loop10
switch-tcl# cli configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.

switch(config-tcl)# cli interface loopback 10
switch(config-if-tcl)# cli description loop10
switch(config-if-tcl)#
```

In an interactive tcl shell, you can also execute Cisco NX-OS commands directly without prepending the tcl **cli** command:

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

switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# description loop10
switch(config-if-tcl)#
```

Tcl Variables

You can use tcl variables as arguments to the Cisco NX-OS commands. You can also pass arguments into tcl scripts. Tcl variables are not persistent.

The following example shows how to use a tcl variable as an argument to a Cisco NX-OS command:

```
switch# tclsh
switch-tcl# set x loop10
switch-tcl# cli "configure terminal ; interface loopback 10 ; description $x"
switch(config-if-tcl)#
```

Tclquit

The **tclquit** command exits the tcl shell regardless of which Cisco NX-OS command mode is currently active. You can also press **Ctrl-C** to exit the tcl shell. The **exit** and **end** commands change Cisco NX-OS command modes. The **exit** command will terminate the tcl shell only from the EXEC command mode.

Tclsh Security

The tcl shell is executed in a sandbox to prevent unauthorized access to certain parts of the Cisco NX-OS system. The system monitors CPU, memory, and file system resources being used by the tcl shell to detect events such as infinite loops, excessive memory utilization, and so on.

You configure the intial tcl environment with the **scripting tcl init** *init-file* command.

You can define the looping limits for the tcl environment with the **scripting tcl recursion-limit** *iterations* command. The default recursion limit is 1000 interations.

Running the tclsh Command

You can run tcl commands from either a script or on the command line using the **tclsh** command.

Note You cannot create a tcl script file at the CLI prompt. You can create the script file on a remote device and copy it to the bootflash: directory on the Cisco NX-OS device.

SUMMARY STEPS

1. **tclsh** [**bootflash:***filename* [*argument* **...**]]

DETAILED STEPS

	Command or Action	Purpose

Step 1	**tclsh** [**bootflash:***filename* [*argument* **...**]]
	Example:
	`switch# tclsh ?`
	`<CR>`
	`bootflash: The file to run`
		Starts a tcl shell.
		If you run the **tclsh** command with no arguments, the shell runs interactively, reading tcl commands from standard input and printing command results and error messages to the standard output. You exit from the interactive tcl shell by typing **tclquit** or **Ctrl-C**.
		If you run the **tclsh** command with arguments, the first argument is the name of a script file containing tcl commands and any additional arguments are made available to the script as variables.

Example

The following example shows an interactive tcl shell:

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# cli show module $x | incl Mod
Mod  Ports  Module-Type                          Model              Status
1    36     36p 40G Ethernet Module             N9k-X9636PQ        ok
Mod  Sw              Hw
Mod  MAC-Address(es)                         Serial-Num

switch-tcl# exit
switch#
```

The following example shows how to run a tcl script:

```
switch# show file bootflash:showmodule.tcl
set x 1
while {$x < 19} {
cli show module $x | incl Mod
set x [expr {$x + 1}]
}

switch# tclsh bootflash:showmodule.tcl
Mod  Ports  Module-Type                          Model              Status
1    36     36p 40G Ethernet Module             N9k-X9636PQ        ok
Mod  Sw              Hw
Mod  MAC-Address(es)                         Serial-Num

switch#
```

Navigating Cisco NX-OS Modes from the tclsh Command

You can change modes in Cisco NX-OS while you are running an interactive tcl shell.

SUMMARY STEPS

1. tclsh
2. configure terminal
3. tclquit

DETAILED STEPS

	Command or Action	Purpose

Step 1	**tclsh**
	Example:
	`switch# tclsh`
	`switch-tcl#`
		Starts an interactive tcl shell.

Step 2	**configure terminal**
	Example:
	`switch-tcl# configure terminal`
	`switch(config-tcl)#`
		Runs a Cisco NX-OS command in the tcl shell, changing modes.
		Note The tcl prompt changes to indicate the Cisco NX-OS command mode.

Step 3	**tclquit**
	Example:
	`switch-tcl# tclquit`
	`switch#`
		Terminates the tcl shell, returning to the starting mode.

Example

The following example shows how to change Cisco NX-OS modes from an interactive tcl shell:

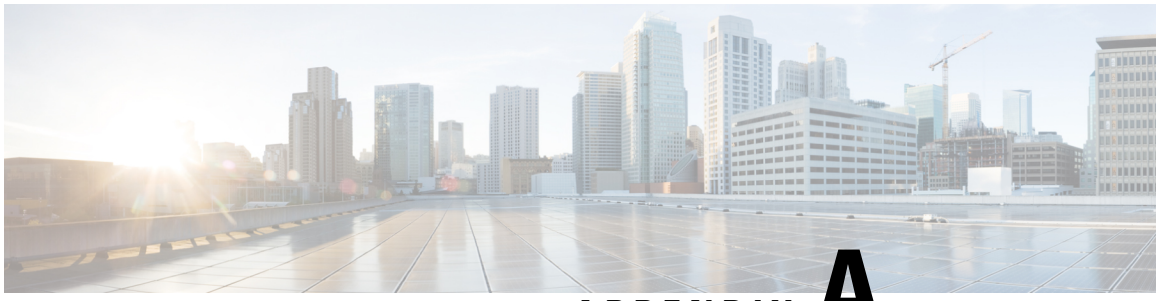
```
switch# tclsh
switch-tcl# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# ?
  description  Enter description of maximum 80 characters
  inherit      Inherit a port-profile
  ip           Configure IP features
  ipv6         Configure IPv6 features
  logging      Configure logging for interface
  no           Negate a command or set its defaults
  rate-limit   Set packet per second rate limit
  shutdown     Enable/disable an interface
  this         Shows info about current object (mode's instance)
  vrf          Configure VRF parameters
  end          Go to exec mode
  exit         Exit from command interpreter
  pop          Pop mode from stack or restore from name
  push         Push current mode to stack or save it under name
  where        Shows the cli context you are in

switch(config-if-tcl)# description loop10
switch(config-if-tcl)# tclquit
```

```
Exiting Tcl
switch#
```

Tcl References

The following titles are provided for your reference:

- Mark Harrison (ed), *Tcl/Tk Tools*, O'Reilly Media, ISBN 1-56592-218-2, 1997
- Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming*, Addison-Wesley, Reading, MA, USA, ISBN 0-201-63474-0, 1998
- John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, USA, ISBN 0-201-63337-X, 1994.
- Brent B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall, Upper Saddle River, NJ, USA, ISBN 0-13-038560-3, 2003.
- J Adrian Zimmer, *Tcl/Tk for Programmers*, IEEE Computer Society, distributed by John Wiley and Sons, ISBN 0-8186-8515-8, 1998.



APPENDIX A

NX-API Response Codes

Table of NX-API Response Codes

The following are the possible NX-API errors, error codes, and messages of an NX-API response.

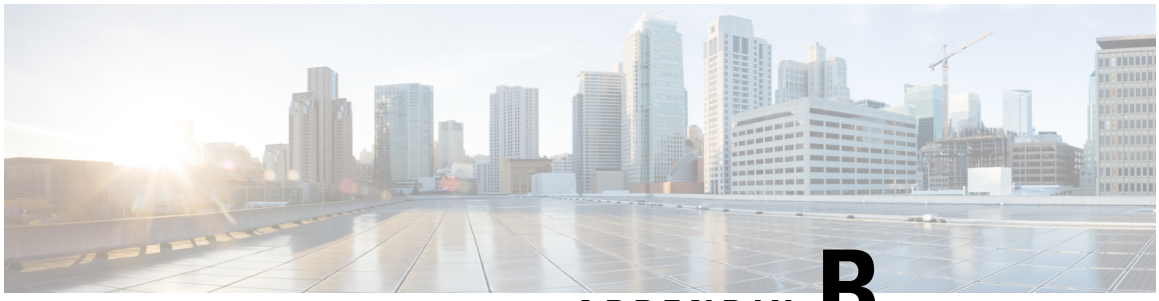
The following are the possible NX-API errors, error codes, and messages of an NX-API response.

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 7: 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.



APPENDIX B

Troubleshooting

About Troubleshooting

Troubleshooting information for Cisco NX-OS programmability is documented in the *Cisco Nexus 9000 Series NX-OS Troubleshooting Guide*.

Troubleshooting