



Cisco Nexus 3400-S Series NX-OS Programmability Guide, Release 9.2(2)

First Published: 2019-07-05

Last Modified: 2019-10-03

Americas Headquarters

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

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

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

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

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

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

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

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

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

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

© 2019 Cisco Systems, Inc. All rights reserved.



CONTENTS

PREFACE

Preface	xiii
Audience	xiii
Document Conventions	xiii
Related Documentation for Cisco Nexus 3000 Series Switches	xiv
Documentation Feedback	xiv
Communications, Services, and Additional Information	xiv

CHAPTER 1

New and Changed Information	1
New and Changed Information	1

CHAPTER 2

Overview	3
Programmability Overview	3
Standard Network Manageability Features	4
Advanced Automation Feature	4
Power on Auto Provisioning Support	4
Programmability Support	4
NX-API Support	4
Python Scripting	5
Tel Scripting	5
Bash	5
Bash Shell Access and Linux Container Support	5
Guest Shell	5
Container Tracker Support	5
Perl Modules	6

PART I

Shells and Scripting	9
-----------------------------	----------

CHAPTER 3**Bash 11**

- About Bash 11
- Guidelines and Limitations 11
- Accessing Bash 12
- Escalate Privileges to Root 13
- Examples of Bash Commands 14
 - Displaying System Statistics 14
 - Running Bash from CLI 15
- Managing Feature RPMs 16
 - RPM Installation Prerequisites 16
 - Installing Feature RPMs from Bash 16
 - Upgrading Feature RPMs 17
 - Downgrading a Feature RPM 18
 - Erasing a Feature RPM 18
- Managing Patch RPMs 18
 - RPM Installation Prerequisites 18
 - Adding Patch RPMs from Bash 19
 - Activating a Patch RPM 21
 - Committing a Patch RPM 22
 - Deactivating a Patch RPM 23
 - Removing a Patch RPM 24
- Persistently Daemonizing an SDK- or ISO-Built Third-Party Process 25
- Persistently Starting Your Application from the Native Bash Shell 26
- Synchronize Files from Active Bootflash to Standby Bootflash 27
- An Example Application in the Native Bash Shell 28

CHAPTER 4**Guest Shell 31**

- About the Guest Shell 31
- Guidelines and Limitations 32
- Accessing the Guest Shell 36
- Resources Used for the Guest Shell 36
- Capabilities in the Guest Shell 37
 - NX-OS CLI in the Guest Shell 37

Network Access in Guest Shell	38
Access to Bootflash in Guest Shell	40
Python in Guest Shell	40
Python 3 in Guest Shell 2.x (Centos 7)	41
Installing RPMs in the Guest Shell	44
Security Posture for Guest Shell	45
Kernel Vulnerability Patches	45
ASLR and X-Space Support	45
Namespace Isolation	45
Root-User Restrictions	46
Resource Management	47
Guest File System Access Restrictions	47
Managing the Guest Shell	48
Disabling the Guest Shell	51
Destroying the Guest Shell	51
Enabling the Guest Shell	52
Replicating the Guest Shell	53
Exporting Guest Shell rootfs	53
Importing Guest Shell rootfs	53
Importing YAML File	55
show guestshell Command	58
Verifying Virtual Service and Guest Shell Information	59
Persistently Starting Your Application from the Guest Shell	60
Procedure for Persistently Starting Your Application from the Guest Shell	61
An Example Application in the Guest Shell	61
Troubleshooting Guest Shell Issues	62

CHAPTER 5**Innovium Shell 65**

About the Innovium Shell	65
Guidelines and Limitations	65
Accessing the Innovium Shell Through CLI API	66
Passing Commands to the Innovium Shell	68

CHAPTER 6**Python API 69**

- About the Python API 69
- Using Python 69
 - Cisco Python Package 69
 - Using the CLI Command APIs 70
 - Invoking the Python Interpreter from the CLI 72
 - Display Formats 72
 - Non-interactive Python 74
 - Running Scripts with Embedded Event Manager 75
 - Python Integration with Cisco NX-OS Network Interfaces 75
 - Cisco NX-OS Security with Python 76
 - Examples of Security and User Authority 76
 - Example of Running Script with Scheduler 78

CHAPTER 7

Scripting with TCL 79

- About Tcl 79
 - Guidelines and Limitations 79
 - Telsh Command Help 79
 - Telsh Command History 80
 - Telsh Tab Completion 80
 - Telsh CLI Command 80
 - Telsh Command Separation 81
 - Tcl Variables 81
 - Tclquit 81
 - Telsh Security 81
- Running the Telsh Command 82
- Navigating Cisco NX-OS Modes from the Telsh Command 82
- Tcl References 83

CHAPTER 8

Kernel Stack 85

- About Kernel Stack 85
- Guidelines and Limitations 85
- Changing the Port Range 86
- Netdevice Property Changes 87

PART II**Applications 89**

CHAPTER 9**Cisco Nexus Application Development - SDK 91**

About the Cisco SDK 91

Installing the SDK 91

Procedure for Installation and Environment Initialization 92

Using the SDK to Build Applications 93

Using RPM to Package an Application 94

Creating an RPM Build Environment 95

Using General RPM Build Procedure 95

Example to Build RPM for collectd with No Optional Plug-Ins 96

Example to Build RPM for collectd with Optional Curl Plug-In 97

CHAPTER 10**NX-SDK 99**

About the NX-SDK 99

Install the NX-SDK 100

Building and Packaging C++ Applications 101

Installing and Running Custom Applications 103

PART III**NX-API 107**

CHAPTER 11**NX-API CLI 109**

About NX-API CLI 109

Transport 109

Message Format 109

Security 110

Using NX-API CLI 110

Escalate Privileges to Root on NX-API 112

Sample NX-API Scripts 113

NX-API Management Commands 114

Working with Interactive Commands Using NX-API 115

NX-API Request Elements 116

NX-API Response Elements 119

Restricting Access to NX-API	120
Updating an iptable	121
Making an Iptable Persistent Across Reloads	122
Table of NX-API Response Codes	123
JSON and XML Structured Output	124
About JSON (JavaScript Object Notation)	125
Examples of XML and JSON Output	125

CHAPTER 12 **NX-API REST** 129

About NX-API REST	129
-------------------	-----

CHAPTER 13 **NX-API Developer Sandbox** 131

About the NX-API Developer Sandbox	131
Guidelines and Limitations	132
Enabling and Accessing the Developer Sandbox	133
Configuring the Message Format and Input Type	133
Using the Developer Sandbox	135
Using the Developer Sandbox to Convert CLI Commands to Payloads	136
Using the Developer Sandbox to Convert from REST Payloads to CLI Commands	139
Using the Developer Sandbox to Convert from RESTCONF to json or XML	143

PART IV **Model-Driven Programmability** 147

CHAPTER 14 **Infrastructure Overview** 149

About Model-Driven Programmability	149
About the Programmable Interface Infrastructure	149

CHAPTER 15 **Managing Components** 153

About the Component RPM Packages	153
Preparing for Installation	155
Downloading Components from the Cisco Artifacts	156
Installing RPM Packages	156
Installing the Programmable Interface Base and Common Model Component RPM Packages	156

CHAPTER 16	OpenConfig YANG	159
	About OpenConfig YANG	159
	Guidelines and Limitations for OpenConfig YANG	159
	Understanding Deletion of BGP Routing Instance	164

CHAPTER 17	NETCONF Agent	167
	About the NETCONF Agent	167
	Guidelines and Limitations	169
	Configuring the NETCONF Agent	169
	Using the NETCONF Agent	170
	Error Messages	173
	Troubleshooting the NETCONF Agent	174

CHAPTER 18	Converting CLI Commands to Network Configuration	175
	Information About XMLIN	175
	Licensing Requirements for XMLIN	175
	Installing and Using the XMLIN Tool	176
	Converting Show Command Output to XML	176
	Configuration Examples for XMLIN	177

CHAPTER 19	RESTCONF Agent	181
	About the RESTConf Agent	181
	Guidelines and Limitations	182
	Configuring the RESTConf Agent	182
	Using the RESTConf Agent	182
	Troubleshooting the RESTConf Agent	183

CHAPTER 20	gRPC Agent	185
	About the gRPC Agent	185
	Guidelines and Limitations	186
	Configuring the gRPC Agent	187
	Using the gRPC Agent	188

Troubleshooting the gRPC Agent 189
 gRPC Protobuf File 189

CHAPTER 21

Dynamic Logger 197
 Prerequisites 197
 Reference 197

CHAPTER 22

Model-Driven Telemetry 205
 About Telemetry 205
 Telemetry Components and Process 205
 High Availability of the Telemetry Process 207
 Licensing Requirements for Telemetry 207
 Installing and Upgrading Telemetry 207
 Guidelines and Limitations 208
 Configuring Telemetry Using the CLI 213
 Configuring Telemetry Using the NX-OS CLI 213
 Configuration Examples for Telemetry Using the CLI 218
 Displaying Telemetry Configuration and Statistics 221
 Displaying Telemetry Log and Trace Information 227
 Configuring Telemetry Using the NX-API 228
 Configuring Telemetry Using the NX-API 228
 Configuration Example for Telemetry Using the NX-API 237
 Telemetry Model in the DME 240

PART V

XML Mangement Interface 243

CHAPTER 23

XML Management Interface 245
 About the XML Management Interface 245
 Information About the XML Management Interface 245
 NETCONF Layers 245
 SSH xmlagent 246
 Licensing Requirements for the XML Management Interface 246
 Prerequisites to Using the XML Management Interface 246
 Using the XML Management Interface 247

Configuring the SSH and the XML Server Options Through the CLI	247
Starting an SSHv2 Session	248
Sending a Hello Message	249
Obtaining XML Schema Definition (XSD) Files	249
Sending an XML Document to the XML Server	250
Creating NETCONF XML Instances	250
Information About Example XML Instances	260
Example XML Instances	260
NETCONF Close Session Instance	260
NETCONF Kill Session Instance	261
NETCONF Copy Config Instance	261
NETCONF Edit Config Instance	262
NETCONF Get Config Instance	263
NETCONF Lock Instance	264
NETCONF Unlock Instance	265
NETCONF Commit Instance: Candidate Configuration Capability	265
NETCONF Confirmed Commit Instance	265
NETCONF Rollback-On-Error Instance	266
NETCONF Validate Capability Instance	266
Additional References	267

APPENDIX A	Streaming Telemetry Sources	269
	About Streaming Telemetry	269
	Guidelines and Limitations	269
	Data Available for Telemetry	269



Preface

This preface includes the following sections:

- [Audience, on page xiii](#)
- [Document Conventions, on page xiii](#)
- [Related Documentation for Cisco Nexus 3000 Series Switches, on page xiv](#)
- [Documentation Feedback, on page xiv](#)
- [Communications, Services, and Additional Information, on page xiv](#)

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.
<i>Italic</i>	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).
[x y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{x y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.

Convention	Description
<i>variable</i>	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

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

Related Documentation for Cisco Nexus 3000 Series Switches

The entire Cisco Nexus 3000 Series switch documentation set is available at the following URL:

<https://www.cisco.com/c/en/us/support/switches/nexus-3000-series-switches/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 nexus3k-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 contains the following topics:

- [New and Changed Information, on page 1](#)

New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 3400-S Series NX-OS Programmability Guide, Release 9.2(2)*.

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
Platform Support	Support added for the Cisco Nexus 3432D-S switch	9.2(2v)	About the Innovium Shell, on page 65 Passing Commands to the Innovium Shell, on page 68
Support for the Cisco Nexus 3400-S	Initial release	9.2(2t)	



CHAPTER 2

Overview

This chapter contains the following topics:

- [Programmability Overview, on page 3](#)
- [Standard Network Manageability Features, on page 4](#)
- [Advanced Automation Feature, on page 4](#)
- [Programmability Support, on page 4](#)

Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 3400-S 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.

For more information on Open NX-OS, see <https://developer.cisco.com/site/nx-os/>.

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 Power On Auto Provisioning (POAP).

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

- Power On Auto Provisioning (POAP) support

Power on Auto Provisioning Support

Power On Auto Provisioning (POAP) automates the process of installing and 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 that are 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.

Programmability Support

Cisco NX-OS on Cisco Nexus 3400-S devices support several capabilities to aid programmability.

NX-API Support

Cisco NX-API allows for HTTP-based programmatic access to the Cisco Nexus 3400-S 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 3400-S platform.

Python Scripting

Cisco Nexus 3400-S devices support Python v2.7.5 and later in both interactive and noninteractive (script) modes.

The Python scripting capability on the devices provides 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 3400-S 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.

Bash

Cisco Nexus 3400-S devices support direct Bourne-Again Shell (Bash) access. With Bash, you can access the underlying Linux system on the device and manage the system.

Bash Shell Access and Linux Container Support

The Cisco Nexus 3400-S switches support direct Linux shell access and Linux containers. With Linux shell access, you can access the underlying Linux system on the Nexus 3400-S switch and manage the underlying system. You can also use Linux containers to securely install your own software and to enhance the capabilities of the Nexus 3400-S switch. For example, you can install bare-metal provisioning tools like Cobbler on a Nexus 3400-S device to enable automatic provisioning of bare-metal servers from the top-of-rack switch.

Guest Shell

The Cisco Nexus 3400-s-Series devices support a guest shell that provides Bash access into a Linux execution space on the host system that is decoupled from the host Cisco NX-OS software. With the guest shell, you can add software packages and update libraries as needed without impacting the host system software.

Container Tracker Support

Cisco NX-OS can communicate with the Kubernetes API Server to understand the capabilities of the containers behind a given switch port.

The following commands communicate with the Kubernetes API Server:

- The **show containers kubernetes** command obtains data from *kube-apiserver* using API calls over HTTP.

- The **kubernetes watch** *resource* command uses a daemon to subscribe to requested resources and process streaming data from *kube-apiserver*.
- The **action** assigned in the **watch** command is performed on pre-defined triggers. (For example, Add or Delete of a Pod.)

Perl Modules

The Cisco Nexus 3400-S switches support the following selected PERL modules.:

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- Class/Struct.pm
- Data/Dumper.pm
- DynaLoader.pm
- Exporter/Heavy.pm
- FileHandle.pm
- File/Basename.pm
- File/Glob.pm
- File/Spec.pm
- File/Spec/Unix.pm
- File/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm

- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- Time/HiRes.pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so



PART I

Shells and Scripting

- [Bash, on page 11](#)
- [Guest Shell, on page 31](#)
- [Innovium Shell, on page 65](#)
- [Python API, on page 69](#)
- [Scripting with TCL, on page 79](#)
- [Kernel Stack, on page 85](#)



CHAPTER 3

Bash

This chapter contains the following topics:

- [About Bash, on page 11](#)
- [Guidelines and Limitations, on page 11](#)
- [Accessing Bash, on page 12](#)
- [Escalate Privileges to Root, on page 13](#)
- [Examples of Bash Commands, on page 14](#)
- [Managing Feature RPMs, on page 16](#)
- [Managing Patch RPMs, on page 18](#)
- [Persistently Daemonizing an SDK- or ISO-Built Third-Party Process, on page 25](#)
- [Persistently Starting Your Application from the Native Bash Shell, on page 26](#)
- [Synchronize Files from Active Bootflash to Standby Bootflash, on page 27](#)
- [An Example Application in the Native Bash Shell, on page 28](#)

About Bash

In addition to the NX-OS CLI, Cisco Nexus 3400-S 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.

Guidelines and Limitations

The Bash shell has the following guidelines and limitations:

- The binaries that are located in the `/isan` folder are meant to be run in an environment which is set up differently from the environment of the shell that is entered from the **run bash** command. It is advisable not to use these binaries from the Bash shell as the behavior within this environment is not predictable.
- When importing Cisco Python modules, do not use Python from the Bash shell. Instead use the more recent Python in NX-OS VSH.
- Some processes and **show** commands can cause a large amount of output. If you are running scripts, and need to terminate long-running output, use Ctrl+C (not Ctrl+Z) to terminate the command output. If you use Ctrl+Z, a SIGCONT (signal continuation) message can be generated, which can cause the script to halt. Scripts that are halted through SIGCONT messages require user intervention to resume operation.

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# configure terminal
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 **run bash** command.

For instance, you can run **whoami** using **run bash** command:

```
run bash whoami
```

You can also run Bash by configuring the user **shelltype**:

```
username foo shelltype bash
```

This command puts you directly into the Bash shell upon login. This does not require **feature bash-shell** to be enabled.

Escalate Privileges to Root

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

The following are guidelines for escalating privileges:

- Only an Admin user can escalate privileges to root.
- Bash must be enabled before escalating privileges.
- Escalation to root is password protected.
- SSH to the switch using `root` username through a non-management interface will default to Linux Bash shell-type access for the root user. Type **vsh** to return to NX-OS shell access.

NX-OS network administrator users must escalate to root to pass configuration commands to the NX-OS VSH if:

- The NX-OS user has a shell-type Bash and logs into the switch with a shell-type Bash.
- The NX-OS user that logged into the switch in Bash continues to use Bash on the switch.

Run **sudo su 'vsh -c "<configuration commands>"** or **sudo bash -c 'vsh -c "<configuration commands>"**.

The following example demonstrates the network-administrator user MyUser with a default shell type Bash using **sudo** to pass configuration commands to the NX-OS.

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ sudo vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show
interface eth1/2 brief"
```

```
-----
Ethernet      VLAN      Type Mode   Status Reason                               Speed   Port
Interface                                           Ch #
-----
Eth1/2        --        eth  routed down  Administratively down             auto(D) --
```

The following example demonstrates the network-administrator user MyUser with default shell type Bash entering the NX-OS and then running Bash on the NX-OS.

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
```

```

Copyright (C) 2002-2019, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.

```

```

switch# run bash
bash-4.2$ vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface
eth1/2 brief"

```

```

-----
Ethernet      VLAN      Type Mode      Status Reason              Speed      Port
Interface                                           Ch #
-----
Eth1/2        --        eth  routed down  Administratively down  auto(D)  --

```



Note Do not use **sudo su -** or the system hangs.

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
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 displays system statistics:

```

switch# run bash
bash-4.2$ cat /proc/meminfo
MemTotal:      32827712 kB
MemFree:       27429772 kB
MemAvailable:  28004236 kB
Buffers:       54296 kB
Cached:        2863648 kB
SwapCached:    0 kB

```

```

Active:          1993452 kB
Inactive:       2616472 kB
Active(anon):   1812124 kB
Inactive(anon): 2192904 kB
Active(file):   181328 kB
Inactive(file): 423568 kB
Unevictable:    0 kB
Mlocked:        0 kB
SwapTotal:      0 kB
SwapFree:       0 kB
Dirty:          24 kB
Writeback:      0 kB
AnonPages:     1691732 kB
Mapped:        578756 kB
Shmem:         2313336 kB
Slab:          248788 kB
SReclaimable:  53660 kB
SUnreclaim:   195128 kB
KernelStack:   11520 kB
PageTables:    58812 kB
NFS_Unstable:  0 kB
Bounce:        0 kB
WritebackTmp:  0 kB
CommitLimit:   16413856 kB
Committed_AS:  23471740 kB
VmallocTotal:  34359738367 kB
VmallocUsed:    579308 kB
VmallocChunk:  34358945788 kB
HardwareCorrupted: 0 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize:  2048 kB
DirectMap4k:   26588 kB
DirectMap2M:   1998848 kB
DirectMap1G:   33554432 kB
bash-4.3#

```

Running Bash from CLI

The following example runs **ps** from Bash using **run bash** 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

```

Managing Feature RPMs

RPM Installation Prerequisites

Use these procedures to verify that the system is ready before installing or adding an RPM.

Procedure

	Command or Action	Purpose
Step 1	switch# show logging logfile grep -i "System ready"	Before running Bash, this step verifies that the system is ready before installing or adding an RPM. Proceed if you see output similar to the following: 2019 Apr 18 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready
Step 2	switch# run bash sudo su Example: switch# run bash sudo su bash-4.2#	Loads Bash.

Installing Feature RPMs from Bash

Procedure

	Command or Action	Purpose
Step 1	sudo yum installed grep platform	Displays a list of the NX-OS feature RPMs installed on the switch.
Step 2	yum list available	Displays a list of the available RPMs.
Step 3	sudo yum -y install rpm	Installs an available RPM.

Example

The following is an example of installing the **bfd** RPM:

```
bash-4.2$ yum list installed | grep n9000
base-files.n9000          3.0.14-r74.2          installed
bfd.lib32_n9000         1.0.0-r0              installed
core.lib32_n9000       1.0.0-r0              installed
eigrp.lib32_n9000     1.0.0-r0              installed
eth.lib32_n9000       1.0.0-r0              installed
```



```

isis.lib32_n9000                1.0.0-r0                installed
lACP.lib32_n9000                1.0.0-r0                installed
linecard.lib32_n9000            1.0.0-r0                installed
lldp.lib32_n9000                1.0.0-r0                installed
ntp.lib32_n9000                 1.0.0-r0                installed
nxos-ssh.lib32_n9000            1.0.0-r0                installed
ospf.lib32_n9000                1.0.0-r0                installed
perf-cisco.n9000_gdb            3.12-r0                 installed
platform.lib32_n9000            1.0.0-r0                installed
shadow-securetty.n9000_gdb      4.1.4.3-r1              installed
snmp.lib32_n9000                1.0.0-r0                installed
svi.lib32_n9000                 1.0.0-r0                installed
sysvinit-inittab.n9000_gdb      2.88dsf-r14             installed
tacacs.lib32_n9000              1.0.0-r0                installed
task-nxos-base.n9000_gdb        1.0-r0                   installed
tor.lib32_n9000                 1.0.0-r0                installed
vtp.lib32_n9000                 1.0.0-r0                installed
bash-4.2$ yum list available
bgp.lib32_n9000                  1.0.0-r0
bash-4.2$ sudo yum -y install bfd

```



Note Upon switch reload during boot up, use the **rpm** command instead of **yum** for persistent RPMs. Otherwise, RPMs initially installed using **yum bash** or **install cli** shows `reponame` or `filename` instead of `installed`.

Upgrading Feature RPMs

Before you begin

There must be a higher version of the RPM in the Yum repository.

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum -y upgrade rpm</code>	Upgrades an installed RPM.

Example

The following is an example of upgrading the **bfd** RPM:

```
bash-4.2$ sudo yum -y upgrade bfd
```

Downgrading a Feature RPM

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum -y downgrade rpm</code>	Downgrades the RPM if any of the Yum repositories has a lower version of the RPM.

Example

The following example shows how to downgrade the **bfd** RPM:

```
bash-4.2$ sudo yum -y downgrade bfd
```

Erasing a Feature RPM



Note The SNMP RPM and the NTP RPM are protected and cannot be erased.

You can upgrade or downgrade these RPMs. It requires a system reload for the upgrade or downgrade to take effect.

For the list of protected RPMs, see `/etc/yum/protected.d/protected_pkgs.conf`.

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum -y erase rpm</code>	Erases the RPM.

Example

The following example shows how to erase the **bfd** RPM:

```
bash-4.2$ sudo yum -y erase bfd
```

Managing Patch RPMs

RPM Installation Prerequisites

Use these procedures to verify that the system is ready before installing or adding an RPM.

Procedure

	Command or Action	Purpose
Step 1	switch# show logging logfile grep -i "System ready"	Before running Bash, this step verifies that the system is ready before installing or adding an RPM. Proceed if you see output similar to the following: 2019 Apr 18 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready
Step 2	switch# run bash sudo su Example: switch# run bash sudo su bash-4.2#	Loads Bash.

Adding Patch RPMs from Bash

Procedure

	Command or Action	Purpose
Step 1	yum list --patch-only	Displays a list of the patch RPMs present on the switch.
Step 2	sudo yum install --add <i>URL_of_patch</i>	Adds the patch to the repository, where <i>URL_of_patch</i> is a well-defined format, such as bootflash:/patch , not in standard Linux format, such as /bootflash/patch .
Step 3	yum list --patch-only available	Displays a list of the patches that are added to the repository but are in an inactive state.

Example

The following is an example of installing the **nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000** RPM:

```
bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B     00:00 ...
patching                   | 951 B     00:00 ...
thirdparty                 | 951 B     00:00 ...
bash-4.2#
bash-4.2# sudo yum install --add bootflash:/nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
```

```

groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B    00:00 ...
patching                   | 951 B    00:00 ...
thirdparty                 | 951 B    00:00 ...
[#####] 70%Install operation 135 completed successfully at Tue Mar 26 17:45:34
2019.

[#####] 100%
bash-4.2#

```

Once the patch RPM is installed, verify that it was installed properly. The following command lists the patches that are added to the repository and are in the inactive state:

```

bash-4.2# yum list --patch-only available
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B    00:00 ...
patching                   | 951 B    00:00 ...
thirdparty                 | 951 B    00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000  1.0.0      patching
bash-4.2#

```

You can also add patches to a repository from a tar file, where the RPMs are bundled in the tar file. The following example shows how to add two RPMs that are part of the nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0.lib32_n9000 tar file to the patch repository:

```

bash-4.2# sudo yum install --add
bootflash:/nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0.lib32_n9000.tar
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B    00:00 ...
patching                   | 951 B    00:00 ...
thirdparty                 | 951 B    00:00 ...
[#####] 70%Install operation 146 completed successfully at Tue Mar 26 21:17:39
2019.

[#####] 100%
bash-4.2#
bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B    00:00 ...
patching                   | 951 B    00:00 ...
patching/primary          | 942 B    00:00 ...
patching                   |          2/2
thirdparty                 | 951 B    00:00 ...
nxos.CSCab00003-n9k_ALL.lib32_n9000  1.0.0      patching
nxos.CSCab00002-n9k_ALL.lib32_n9000  1.0.0      patching
bash-4.2#

```

Activating a Patch RPM

Before you begin

Verify that you have added the necessary patch RPM to the repository using the instructions in [Adding Patch RPMs from Bash](#), on page 19.

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum install patch_RPM --nocommit</code>	<p>Activates the patch RPM, where <i>patch_RPM</i> is a patch that is located in the repository. Do not provide a location for the patch in this step.</p> <p>Note Adding the <code>--nocommit</code> flag to the command means that the patch RPM is activated in this step, but not committed. See Committing a Patch RPM, on page 22 for instructions on committing the patch RPM after you have activated it.</p>

Example

The following example shows how to activate the `nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000` patch RPM:

```
bash-4.2# sudo yum install nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000 --nocommit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                               | 1.1 kB    00:00 ...
localdb                                    | 951 B    00:00 ...
patching                                   | 951 B    00:00 ...
thirdparty                                 | 951 B    00:00 ...
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                                Arch      Version      Repository    Size
=====
Installing:
nxos.CSCab00001-n9k_ALL                lib32_n9000 1.0.0        patching      28 k

Transaction Summary
=====
Install      1 Package

Total download size: 28 k
Installed size: 82 k
Is this ok [y/N]: y
```

```

Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000          1/1
[#####] 90%error: reading
/var/sysmgr/tmp/patches/CSCab00001-n9k_ALL/isan/bin/sysinfo manifest, non-printable characters
found

Installed:
  nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0

Complete!
Install operation 140 completed successfully at Tue Mar 27 18:07:40 2018.

[#####] 100%
bash-4.2#

```

Enter the following command to verify that the patch RPM was activated successfully:

```

bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages

groups-repo                | 1.1 kB      00:00 ...
localdb                    | 951 B      00:00 ...
patching                   | 951 B      00:00 ...
thirdparty                 | 951 B      00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000  1.0.0      installed
bash-4.2#

```

Committing a Patch RPM

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum install <i>patch_RPM</i> --commit</code>	Commits the patch RPM. The patch RPM must be committed to keep it active after reloads.

Example

The following example shows how to commit the `nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000` patch RPM:

```

bash-4.2# sudo yum install nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000 --commit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages

groups-repo                | 1.1 kB      00:00 ...
localdb                    | 951 B      00:00 ...
patching                   | 951 B      00:00 ...
thirdparty                 | 951 B      00:00 ...
Install operation 142 completed successfully at Tue Mar 27 18:13:16 2018.

[#####] 100%
bash-4.2#

```

Enter the following command to verify that the patch RPM was committed successfully:

```
bash-4.2# yum list --patch-only committed
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                               | 1.1 kB      00:00 ...
localdb                                    | 951 B       00:00 ...
patching                                   | 951 B       00:00 ...
thirdparty                                 | 951 B       00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000      1.0.0        installed
bash-4.2#
```

Deactivating a Patch RPM

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum erase <i>patch_RPM</i> --nocommit</code>	Deactivates the patch RPM. Note Adding the <code>--nocommit</code> flag to the command means that the patch RPM is only deactivated in this step.
Step 2	<code>sudo yum install <i>patch_RPM</i> --commit</code>	Commits the patch RPM. You will get an error message if you try to remove the patch RPM without first committing it.

Example

The following example shows how to deactivate the `nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000` patch RPM:

```
bash-4.2# sudo yum erase nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000 --nocommit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0 will be erased
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch           Version         Repository      Size
=====
Removing:
nxos.CSCab00001-n9k_ALL lib32_n9000    1.0.0           @patching      82 k

Transaction Summary
=====
Remove                1 Package

Installed size: 82 k
Is this ok [y/N]: y
Downloading Packages:
```

```

Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
[#####          ] 30%error: reading
/var/sysmgr/tmp/patches/CSCab00001-n9k_ALL/isan/bin/sysinfo manifest, non-printable characters
found
Erasing      : nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000          1/1
[#####          ] 90%
Removed:
  nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0

Complete!
Install operation 143 completed successfully at Tue Mar 27 21:03:47 2018.

[#####          ] 100%
bash-4.2#

```

You must commit the patch RPM after deactivating it. If you do not commit the patch RPM after deactivating it, you will get an error message if you try to remove the patch RPM using the instructions in [Removing a Patch RPM, on page 24](#).

```

bash-4.2# sudo yum install nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000 --commit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo          | 1.1 kB    00:00 ...
localdb              | 951 B    00:00 ...
patching             | 951 B    00:00 ...
thirdparty          | 951 B    00:00 ...
Install operation 144 completed successfully at Tue Mar 27 21:09:28 2018.

[#####          ] 100%
bash-4.2#

```

Enter the following command to verify that the patch RPM has been committed successfully:

```

bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo          | 1.1 kB    00:00 ...
localdb              | 951 B    00:00 ...
patching             | 951 B    00:00 ...
thirdparty          | 951 B    00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000  1.0.0      patching
bash-4.2#

```

Removing a Patch RPM

Procedure

	Command or Action	Purpose
Step 1	<code>sudo yum install --remove <i>patch_RPM</i></code>	Removes an inactive patch RPM.

Example

The following example shows how to remove the `nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000` patch RPM:

```
bash-4.2# sudo yum install --remove nxos.CSCab00001-n9k_ALL-1.0.0.lib32_n9000
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB      00:00 ...
localdb                    | 951 B       00:00 ...
patching                   | 951 B       00:00 ...
thirdparty                 | 951 B       00:00 ...
[#####          ] 50%Install operation 145 completed successfully at Tue Mar 27 21:11:05
 2018.

[#####] 100%
bash-4.2#
```



Note If you see the following error message after attempting to remove the patch RPM:

Install operation 11 "failed because patch was not committed". at Wed Mar 28 22:14:05 2018

Then you did not commit the patch RPM before attempting to remove it. See [Deactivating a Patch RPM, on page 23](#) for instructions on committing the patch RPM before attempting to remove it.

Enter the following command to verify that the inactive patch RPM was removed successfully:

```
bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                | 1.1 kB      00:00 ...
localdb                    | 951 B       00:00 ...
patching                   | 951 B       00:00 ...
patching/primary          | 197 B       00:00 ...
thirdparty                 | 951 B       00:00 ...
bash-4.2#
```

Persistently Daemonizing an SDK- or ISO-Built Third-Party Process

Your application should have a startup Bash script that gets installed in `/etc/init.d/application_name`. This startup Bash script should have the following general format. For more information about this format, see <http://linux.die.net/man/8/chkconfig>.

```
#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
```

```

# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL

```

Persistently Starting Your Application from the Native Bash Shell

Procedure

-
- Step 1** Install your application startup Bash script that you created into `/etc/init.d/application_name`
 - Step 2** Start your application with `/etc/init.d/application_name start`
 - Step 3** Enter `chkconfig --add application_name`
 - Step 4** Enter `chkconfig --level 3 application_name on`
Run level 3 is the standard multi-user run level, and the level at which the switch normally runs.
 - Step 5** Verify that your application is scheduled to run on level 3 by running `chkconfig --list application_name` and confirm that level 3 is set to on
 - Step 6** Verify that your application is listed in `/etc/rc3.d`. You should see something like this, where there is an 'S' followed by a number, followed by your application name (`tcollector` in this example), and a link to your Bash startup script in `../init.d/application_name`
-

```
bash-4.2# ls -l /etc/rc3.d/tcollector
```

```
lrwxrwxrwx 1 root root 20 Sep 25 22:56 /etc/rc3.d/S15tcollector -> ../init.d/tcollector
```

```
bash-4.2#
```

Synchronize Files from Active Bootflash to Standby Bootflash

Cisco Nexus 3400-S switches are generally configured with two supervisor modules to provide high availability (one active supervisor module and one standby supervisor module). Each supervisor module has its own bootflash file system for file storage, and the Active and Standby bootflash file systems are generally independent of each other. If there is a need for specific content on the active bootflash, that same content is probably also needed on the standby bootflash in case there is a switchover at some point.

Certain files and directories on the active supervisor module, or active bootflash (`/bootflash`), can be automatically synchronized to the standby supervisor module, or standby bootflash (`/bootflash_sup-remote`), if the standby supervisor module is up and available. You can select the files and directories to be synchronized by loading Bash on your switch, then adding the files and directories that you want to have synchronized from the active bootflash to the standby bootflash into the editable file `/bootflash/bootflash_sync_list`.

For example:

```
switch# run bash
bash-4.2# echo "/bootflash/home/admin" | sudo tee --append /bootflash/bootflash_sync_list
bash-4.2# echo "/bootflash/nxos.5.bin" | sudo tee --append /bootflash/bootflash_sync_list
bash-4.2# cat /bootflash/bootflash_sync_list
/bootflash/home/admin
/bootflash/nxos.5.bin
```

When changes are made to the files or directories on the active bootflash, these changes are automatically synchronized to standby bootflash, if the standby bootflash is up and available. If the standby bootflash is rebooted, either as a regular boot, switchover or manual standby reload, a catch-up synchronization of changes to the active bootflash is pushed out to the standby bootflash, once the standby supervisor comes online.

Following are the characteristics and restrictions for the editable `/bootflash/bootflash_sync_list` file:

- The `/bootflash/bootflash_sync_list` file is automatically created on the first run and is empty at that initial creation state.
- Entries in the `/bootflash/bootflash_sync_list` file follow these guidelines:
 - One entry per line
 - Entries are given as Linux paths (for example, `/bootflash/img.bin`)
 - Entries must be within the `/bootflash` file system
- The `/bootflash/bootflash_sync_list` file itself is automatically synchronized to the standby bootflash. You can also manually copy the `/bootflash/bootflash_sync_list` file to or from the supervisor module using the **copy** virtual shell (VSH) command.
- You can edit the `/bootflash/bootflash_sync_list` file directly on the supervisor module with the following command:

```
run bash vi /bootflash/bootflash_sync_list
```

All output from the synchronization event is redirected to the log file `/var/tmp/bootflash_sync.log`. You can view or tail this log file using either of the following commands:

```
run bash less /var/tmp/bootflash_sync.log
```

```
run bash tail -f /var/tmp/bootflash_sync.log
```

The synchronization script will not delete files from the standby bootflash directories unless it explicitly receives a delete event for the corresponding file on the active bootflash directories. Sometimes, the standby bootflash might have more used space than the active bootflash, which results in the standby bootflash running out of space when the active bootflash is synchronizing to it. To make the standby bootflash an exact mirror of the active bootflash (to delete any extra files on the standby bootflash), enter the following command:

```
run bash sudo rsync -a --delete /bootflash/ /bootflash_sup-remote/
```

The synchronization script should continue to run in the background without crashing or exiting. However, if it does stop running for some reason, you can manually restart it using the following command:

```
run bash sudo /isan/etc/rc.d/rc.isan-start/S98bootflash_sync.sh start
```

An Example Application in the Native Bash Shell

The following example demonstrates an application in the Native Bash Shell:

```
bash-4.2# cat /etc/init.d/hello.sh
#!/bin/bash

PIDFILE=/tmp/hello.pid
OUTPUTFILE=/tmp/hello

echo $$ > $PIDFILE
rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
bash-4.2#
bash-4.2#
bash-4.2# cat /etc/init.d/hello
#!/bin/bash
#
# hello Trivial "hello world" example Third Party App
#
# chkconfig: 2345 15 85
# description: Trivial example Third Party App
#
### BEGIN INIT INFO
# Provides: hello
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Trivial example Third Party App
### END INIT INFO

PIDFILE=/tmp/hello.pid

# See how we were called.
case "$1" in
start)
    /etc/init.d/hello.sh &
```

```

    RETVAL=$?
;;
stop)
    kill -9 `cat $PIDFILE`
    RETVAL=$?
;;
status)
    ps -p `cat $PIDFILE`
    RETVAL=$?
;;
restart|force-reload|reload)
    kill -9 `cat $PIDFILE`
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
bash-4.2#
bash-4.2# chkconfig --add hello
bash-4.2# chkconfig --level 3 hello on
bash-4.2# chkconfig --list hello
hello          0:off  1:off  2:on   3:on   4:on   5:on   6:off
bash-4.2# ls -al /etc/rc3.d/*hello*
lrwxrwxrwx 1 root root 15 Sep 27 18:00 /etc/rc3.d/S15hello -> ../init.d/hello
bash-4.2#
bash-4.2# reboot

```

After reload

```

bash-4.2# ps -ef | grep hello
root      8790      1  0 18:03 ?        00:00:00 /bin/bash /etc/init.d/hello.sh
root      8973    8775  0 18:04 ttyS0    00:00:00 grep hello
bash-4.2#
bash-4.2# ls -al /tmp/hello*
-rw-rw-rw- 1 root root 205 Sep 27 18:04 /tmp/hello
-rw-rw-rw- 1 root root   5 Sep 27 18:03 /tmp/hello.pid
bash-4.2# cat /tmp/hello.pid
8790
bash-4.2# cat /tmp/hello
Sun Sep 27 18:03:49 UTC 2015
Hello World
Sun Sep 27 18:03:59 UTC 2015
Hello World
Sun Sep 27 18:04:09 UTC 2015
Hello World
Sun Sep 27 18:04:19 UTC 2015
Hello World
Sun Sep 27 18:04:29 UTC 2015
Hello World
Sun Sep 27 18:04:39 UTC 2015
Hello World
bash-4.2#

```




CHAPTER 4

Guest Shell

This chapter contains the following topics:

- [About the Guest Shell, on page 31](#)
- [Guidelines and Limitations, on page 32](#)
- [Accessing the Guest Shell, on page 36](#)
- [Resources Used for the Guest Shell, on page 36](#)
- [Capabilities in the Guest Shell, on page 37](#)
- [Security Posture for Guest Shell, on page 45](#)
- [Managing the Guest Shell, on page 48](#)

About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, the Cisco Nexus 3400-S 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 over Linux network interfaces.
- Access to Cisco Nexus 3400-S bootflash.
- Access to Cisco Nexus 3400-S volatile tmpfs.
- Access to Cisco Nexus 3400-S CLI.
- Access to Cisco NX-API REST.
- The ability to install and run Python scripts.
- The ability to install and run 32-bit and 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 appears in the **show virtual-service** command output.



Note By default, the Guest Shell occupies approximately 35 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.

Guidelines and Limitations

Common Guidelines Across All Releases



Important If you have performed custom work inside your installation of the Guest Shell, save your changes to bootflash, off-box storage, or elsewhere outside the Guest Shell root file system before performing a `guestshell upgrade`.

The `guestshell upgrade` command essentially performs a `guestshell destroy` and `guestshell enable` in succession.

- Use the `run guestshell` CLI command to access the Guest Shell on the Cisco Nexus device: The `run guestshell` command parallels the `run bash` command that is used to access the host shell. This command allows you to access the Guest Shell and get a Bash prompt or run a command within the context of the Guest Shell. The command uses password-less SSH to an available port on the localhost in the default network namespace.
- The `sshd` utility can secure the pre-configured SSH access into the Guest Shell by listening on localhost to avoid connection attempts from outside the network. The `sshd` utility has the following features:
 - It is configured for key-based authentication without fallback to passwords.
 - Only `root` can read keys use to access the Guest Shell after Guest Shell restarts.
 - Only `root` can read the file that contains the key on the host to prevent a non-privileged user with host Bash access from being able to use the key to connect to the Guest Shell. Network-admin users may start another instance of `sshd` in the Guest Shell to allow remote access directly into the Guest Shell, but any user that logs into the Guest Shell is also given network-admin privilege.



Note Introduced in Guest Shell 2.2 (0.2), the key file is readable for whom the user account was created for.

In addition, the Guest Shell accounts are not automatically removed, and must be removed by the network administrator when no longer needed.

Guest Shell installations before 2.2 (0.2) will not dynamically create individual user accounts.

- Installing the Cisco Nexus series switch software release on a fresh out-of-the-box Cisco Nexus switch will automatically enable the Guest Shell. Subsequent upgrades to the Cisco Nexus series switch software will not automatically upgrade Guest Shell.

- Guest Shell releases increment the major number when distributions or distribution versions change.
- Guest Shell releases increment the minor number when CVEs have been addressed. The Guest Shell will update CVEs only when CentOS makes them publically available.
- Cisco recommends using **yum update** to pick up third-party security vulnerability fixes directly from the CentOS repository. This provides the flexibility of getting updates as, and when, available without needing to wait for a Cisco NX-OS software update.

Alternatively, using the **guestshell update** command would replace the existing Guest Shell rootfs. Any customizations and software package installations would then need to be performed again within the context of this new Guest Shell rootfs.

Upgrading from Guest Shell 1.0 to Guest Shell 2.x

Guest Shell 2.x is based upon a CentOS 7 root file system. If you have an off-box repository of `.conf` files or utilities that pulled the content down into Guest Shell 1.0, you will need to repeat the same deployment steps in Guest Shell 2.x. Your deployment script may need to be adjusted to account for the CentOS 7 differences.

Guest Shell 2.x

The Cisco NX-OS automatically installs and enables the Guest Shell by default on systems with sufficient resources. However, if the device is reloaded with a Cisco NX-OS image that does not provide Guest Shell support, the installer will automatically remove the existing Guest Shell and issue a `%VMAN-2-INVALID_PACKAGE`.



Note Systems with 4GB of RAM will not enable Guest Shell by default. Use the **guestshell enable** command to install and enable Guest Shell.

The **install all** command validates the compatibility between the current Cisco NX-OS image against the target Cisco NX-OS image.

The following is an example output from installing an incompatible image:

```
switch#
Installer will perform compatibility check first. Please wait.
uri is: /
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 "" version info using image bootflash:/.
[#####] 100% -- SUCCESS
Preparing "bios" version info using image bootflash:/.
[#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[#####] 100% -- SUCCESS
Preparing "nxos" version info using image bootflash:/.
[#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[#####] 100% -- SUCCESS
```

```

Preparing "" version info using image bootflash:/.
[#####] 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#

```



Note As a best practice, remove the Guest Shell with the **guestshell destroy** command before reloading an older Cisco Nexus image that does not support the Guest Shell.

Pre-Configured SSHD Service

The Guest Shell starts an OpenSSH server upon boot up. The server listens on a randomly generated port 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 virtual-shell when the guestshell keyword is entered. If this server is killed or its configuration (residing in `/etc/ssh/sshd_config-cisco`) is altered, access to the Guest Shell from the NX-OS CLI might not work.

The following steps instantiate an OpenSSH server within the Guest Shell as root:

1. Determine which network namespace or VRF you want to establish your SSH connections through.
2. Determine the port that you want OpenSSH to listen on. Use the NX-OS command **show socket connection** to view ports already in use.



Note The Guest Shell sshd service for password-less access uses a randomized port from 17680 through 49150. To avoid port conflict, choose a port outside this range.

The following steps start the OpenSSH server. The examples start the OpenSSH server for management netns on IP address 10.122.84.34:2222:

1. Create the following files: `/usr/lib/systemd/system/sshd-mgmt.service` and `/etc/ssh/sshd-mgmt_config`. The files should have the following configurations:

```

-rw-r--r-- 1 root root 394 Apr 7 14:21 /usr/lib/systemd/system/sshd-mgmt.service
-rw----- 1 root root 4478 Apr 7 14:22 /etc/ssh/sshd-mgmt_config

```

2. Copy the Unit and Service contents from the `/usr/lib/systemd/system/ssh.service` file to `sshd-mgmt.service`.

```

[Unit]
Description=OpenSSH server daemon

```

```

After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/ssh
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/sbin/ip netns exec management /usr/sbin/sshd -f /etc/ssh/sshd-mgmt_config
-D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target

```

3. Copy the contents of `/etc/ssh/sshd-config` to `/etc/ssh/sshd-mgmt_config`. Modify the ListenAddress IP and port as necessary.

```

Port 2222
ListenAddress 10.122.84.34

```

4. Start the `systemctl` daemon using the following commands:

```

sudo systemctl daemon-reload
sudo systemctl start sshd-mgmt.service
sudo systemctl status sshd-mgmt.service -l

```

5. (optional) Check the configuration.

```

ss -tnldp | grep 2222

```

6. SSH into Guest Shell:

```

ssh -p 2222 guestshell@10.122.84.34

```

7. Save the configuration across multiple Guest Shell or switch reboots.

```

sudo systemctl enable sshd-mgmt.service

```

8. For passwordless SSH or SCP and remote execution, generate the public and private keys for the user ID you want to user for SSH/SCP using the `ssh-keygen -t dsa` command.

The key is then stored in the `id_rsa` and `id_rsa.pub` files in the `/.ssh` directory:

```

[root@node01 ~]# cd ~/.ssh
[root@node02 .ssh]# ls -l
total 8
-rw-----. 1 root root 1675 May 5 15:01 id_rsa
-rw-r--r--. 1 root root 406 May 5 15:01 id_rsa.pub

```

9. Copy the public key into the machine you want to SSH into and fix permissions:

```

cat id_rsa.pub >> /root/.ssh/authorized_keys
chmod 700 /root/.ssh
chmod 600 /root/.ssh/*

```

10. SSH or SCP into the remote switch without a password:

```

ssh -p <port#> userid@hostname [<remote command>]
scp -P <port#> userid@hostname/filepath /destination

```

localtime

The Guest Shell shares `/etc/localtime` with the host system.



Note If you do not want to share the same localtime with the host, this symlink can be broken and a Guest Shell specific `/etc/localtime` can be created.

```
switch(config)# clock timezone PDT -7 0
switch(config)# clock set 10:00:00 27 Jan 2017
Fri Jan 27 10:00:00 PDT 2017
switch(config)# show clock
10:00:07.554 PDT Fri Jan 27 2017
switch(config)# run guestshell
guestshell:~$ date
Fri Jan 27 10:00:12 PDT 2017
```

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.



Note The Guest Shell is automatically enabled on systems with more than 4 GB of RAM.

```
switch# run guestshell ls -al /bootflash/*.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.



Note In version 2.2(0.2) and later, the Guest Shell dynamically creates user accounts with the same information as the user logged into switch. However, all other information is not shared between the switch and the Guest Shell user accounts.

In addition, the Guest Shell accounts are not automatically removed, and must be removed by the network administrator when no longer needed.

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 more 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	200MB	200/2000MB

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.

Capabilities in the Guest Shell

The Guest Shell has several utilities and capabilities available by default.

The Guest Shell runs CentOS 7 Linux which supports Yum install software packages that are built for this distribution. The Guest Shell is pre-populated with many of the common tools that you would expect on a networking device including **net-tools**, **iproute**, **tcpdump** and OpenSSH. Python 2.7.5 and later is included by default as is the PIP for installing extra Python packages.

By default the Guest Shell is a 64-bit execution space. If the switch needs 32-bit support, you can install the `glibc.i686` package.

The Guest Shell has access to the Linux network interfaces used to represent the management and data ports of the switch. Typical Linux methods and utilities like **ifconfig** and **ethtool** can be used to collect counters. When an interface is placed into a VRF in the NX-OS CLI, the Linux network interface is placed into a network namespace for that VRF. The name spaces can be seen at `/var/run/netns` and the **ip netns** utility can be used to run in the context of different namespaces. A couple of utilities, **chvrf** and **vrinfo**, are provided for running in a different namespace and getting information about which namespace or VRF a process is running in.

The switch's systemd manages services in CentOS 7 environments, including the Guest Shell.

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 may be in single or double quotes:

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

The NX-OS CLI can be chained together:

```
[guestshell@guestshell ~]$ dohost "sh lldp time | in Hold" "show cdp global"
```

```

Holdtime in seconds: 120
Global CDP information:
CDP enabled globally
Refresh time is 21 seconds
Hold time is 180 seconds
CDPv2 advertisements is enabled
DeviceID TLV in System-Name(Default) Format
[guestshell@guestshell ~]$

```

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@guestshell ~]$ dohost "conf t ; cdp timer 13 ; show run | inc cdp"
Enter configuration commands, one per line. End with CNTL/Z.
cdp timer 13
[guestshell@guestshell ~]$

```



Note With Guest Shell version 2.2 (0.2) and later, commands that are issued on the host through the **dohost** command are run with privileges that are based on the effective role of the Guest Shell user.

Prior versions of Guest Shell run commands with network-admin level privileges.

The **dohost** command fails when the number of UDS connections to NX-API are at the maximum allowed.

Network Access in Guest Shell

The NX-OS switch ports are represented in the Guest Shell as Linux network interfaces. Typical Linux methods like view stats in `/proc/net/dev` through `ifconfig`, or `ethtool` are all supported:

The Guest Shell has several typical network utilities included by default. They can be used on different VRFs using the **chvrf vrf command** command.

```

[guestshell@guestshell bootflash]$ ifconfig Eth1-47
Eth1-47: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 13.0.0.47 netmask 255.255.255.0 broadcast 13.0.0.255
ether 54:7f:ee:8e:27:bc txqueuelen 100 (Ethernet)
RX packets 311442 bytes 21703008 (20.6 MiB)
RX errors 0 dropped 185 overruns 0 frame 0
TX packets 12967 bytes 3023575 (2.8 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Within the Guest Shell, the networking state can be monitored, but not changed. To change networking state, use the NX-OS CLI or the appropriate Linux utilities in the host Bash shell.

The **tcpdump** command is packaged with the Guest Shell to allow packet tracing of punted traffic on the management or switch ports.

The **sudo ip netns exec management ping 10.28.38.48** utility is a common method for running a command in the context of a specified network namespace. This can be done within the Guest Shell:

```

[guestshell@guestshell bootflash]$ sudo ip netns exec management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms

```

The `chvrf` utility is provided as a convenience:

```
guestshell@guestshell bootflash]$ chvrf management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms
```



Note Commands that are run without the `chvrf` command are run in the current VRF or network namespace.

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@guestshell ~]$ cd /bootflash
[guestshell@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@guestshell bootflash]$ ls -al index.html
-rw-r--r-- 1 guestshe users 1804 Sep 13 20:28 index.html
[guestshell@guestshell bootflash]$
[guestshell@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@guestshell bootflash]$
```

To obtain a list of VRFs on the system, use the `show vrf` command natively from NX-OS or through the `dohost` command:

Example:

```
[guestshell@guestshell bootflash]$ dohost 'sh vrf'
VRF-Name  VRF-ID  State  Reason
default   1        Up     --
management 2        Up     --
red       6        Up     --
```

Within the Guest Shell, the network namespaces associated with the VRFs are what are actually used. It can be more convenient to just see which network namespaces are present:

```
[guestshell@guestshell bootflash]$ ls /var/run/netns
default management red
[guestshell@guestshell bootflash]$
```

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 name server and domain as appropriate for the network.

Example:

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

The name server 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 3400-S 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"
```



Note While the name of the user within the Guest Shell is the same as when on the host, the Guest Shell is in a separate user namespace, and the UID does not match that of the user on the host. The file permissions for group and others control the type of access the Guest Shell user has on the file.

Python in Guest Shell

Python can be used interactively or Python scripts can be run in the Guest Shell.

Example:

```
guestshell:~$ python
python
Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2
```



```
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
guestshell:~$
```

The pip Python package manager is included in the Guest Shell to allow the network-admin to install new Python packages.

Example:

```
[guestshell@guestshell ~]$ sudo su
[root@guestshell guestshell]# pip install Markdown
Collecting Markdown
Downloading Markdown-2.6.2-py2.py3-none-any.whl (157kB)
100% |#####| 159kB 1.8MB/s
Installing collected packages: Markdown
Successfully installed Markdown-2.6.2
[root@guestshell guestshell]# pip list | grep Markdown
Markdown (2.6.2)
[root@guestshell guestshell]#
```



Note You must enter the **sudo su** command before entering the **pip install** command.

Python 3 in Guest Shell 2.x (Centos 7)

Guest Shell 2.X provides a Centos 7.1 environment, which does not have Python 3 installed by default. There are multiple methods of installing Python 3 on Centos 7.1, such as using third-party repositories or building from source. Another option is using the Red Hat Software Collections, which supports installing multiple versions of Python within the same system.

To install the Red Hat Software Collections (SCL) tool:

1. Install the scl-utils package.
2. Enable the Centos SCL repository and install one of its provided Python 3 RPMs.

```
[admin@guestshell ~]$ sudo su
[root@guestshell admin]# yum install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
  Installing : scl-utils-20130529-19.e17.x86_64                1/1
  Verifying  : scl-utils-20130529-19.e17.x86_64                1/1

Installed:
  scl-utils.x86_64 0:20130529-19.e17

Complete!

[root@guestshell admin]# yum install -y centos-release-scl | tail
  Verifying : centos-release-scl-2-3.e17.centos.noarch          1/2
  Verifying : centos-release-scl-rh-2-3.e17.centos.noarch       2/2

Installed:
  centos-release-scl.noarch 0:2-3.e17.centos

Dependency Installed:
```

```
centos-release-scl-rh.noarch 0:2-3.el7.centos
```

Complete!

```
[root@guestshell admin]# yum install -y rh-python36 | tail
warning: /var/cache/yum/x86_64/7/centos-scl-rh/packages/rh-python36-2.0-1.el7.x86_64.rpm:
Header V4 RSA/SHA1 Signature, key ID f2ee9d55: NOKEY
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
[Errno 12] Timeout on
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm: (28,
'Operation too slow. Less than 1000 bytes/sec transferred the last 30 seconds')
Trying other mirror.
Importing GPG key 0xF2EE9D55:
  Userid      : "CentOS SoftwareCollections SIG
(https://wiki.centos.org/SpecialInterestGroup/SCLo) <security@centos.org>"
Fingerprint: c4db d535 blfb ba14 f8ba 64a8 4eb8 4e71 f2ee 9d55
Package      : centos-release-scl-rh-2-3.el7.centos.noarch (@extras)
From         : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-SIG-SCLo
rh-python36-python-libs.x86_64 0:3.6.9-2.el7
rh-python36-python-pip.noarch 0:9.0.1-2.el7
rh-python36-python-setuptools.noarch 0:36.5.0-1.el7
rh-python36-python-virtualenv.noarch 0:15.1.0-2.el7
rh-python36-runtime.x86_64 0:2.0-1.el7
scl-utils-build.x86_64 0:20130529-19.el7
xml-common.noarch 0:0.6.3-39.el7
zip.x86_64 0:3.0-11.el7
```

Complete!

Using SCL, it is possible to create an interactive bash session with Python 3's environment variables automatically setup.



Note The root user is not needed to use the SCL Python installation.

```
[admin@guestshell ~]$ scl enable rh-python36 bash
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The Python SCL installation also provides the pip utility.

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/1d/23c926cc841ea67cd02a00aba99ae0f828e89d72b2190f27c11d457fb/requests-2.22.0-py2.py3-none-any.whl
(57kB)
  100% |#####| 61kB 211kB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading
https://files.pythonhosted.org/packages/14/2c/cd551d81d8e15200e1cf41cd03869a46fe7226e7450af7a6545b0fc474c9/idna-2.8-py2.py3-none-any.whl
(58kB)
  100% |#####| 61kB 279kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading
https://files.pythonhosted.org/packages/cc/a9/01ffbf562e4274b6487b4bb1dbbc7ca55c7510b22e4c51f1409844368/chardet-3.0.4-py2.py3-none-any.whl
(133kB)
  100% |#####| 143kB 441kB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading
https://files.pythonhosted.org/packages/b9/63/d50cac98a0fb006c55a3993bfd9d9a765a24de7890bc9cf5db9e99/certifi-2019.11.28-py2.py3-none-any.whl
```

```
(156kB)
100% |#####| 163kB 447kB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests)
  Downloading
https://files.pythonhosted.org/packages/e8/74/6e4f91745020f967d09332b2b289d10090957334692ab88a4afe91b77f/urllib3-1.25.8-py2.py3-none-any.whl
(125kB)
100% |#####| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>
```

The default Python 2 installation can be used alongside the SCL Python installation.

```
[admin@guestshell ~]$ which python3
/opt/rh/rh-python36/root/usr/bin/python3
[admin@guestshell ~]$ which python2
/bin/python2
[admin@guestshell ~]$ python2
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello world!'
Hello world!
```

Software Collections makes it possible to install multiple versions of the same RPM on a system. In this case, it is possible to install Python 3.5 in addition to Python 3.6.

```
[admin@guestshell ~]$ sudo yum install -y rh-python35 | tail
Dependency Installed:
rh-python35-python.x86_64 0:3.5.1-13.e17
rh-python35-python-devel.x86_64 0:3.5.1-13.e17
rh-python35-python-libs.x86_64 0:3.5.1-13.e17
rh-python35-python-pip.noarch 0:7.1.0-2.e17
rh-python35-python-setuptools.noarch 0:18.0.1-2.e17
rh-python35-python-virtualenv.noarch 0:13.1.2-2.e17
rh-python35-runtime.x86_64 0:2.0-2.e17
```

Complete!

```
[admin@guestshell ~]$ scl enable rh-python35 python3
Python 3.5.1 (default, May 29 2019, 15:41:33)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Note Creating new interactive bash sessions when multiple Python versions are installed in SCL can cause an issue where the libpython shared object file cannot be loaded. There is a workaround where you can use the **source scl_source enable python-installation** command to properly set up the environment in the current bash session.

The default Guest Shell storage capacity is not sufficient to install Python 3. Use the **guestshell resize rootfs size-in-MB** command to increase the size of the file system. Typically, setting the rootfs size to 550 MB is sufficient.

Installing RPMs in the Guest Shell

The `/etc/yum/repos.d/CentOS-Base.repo` file is set up to use the CentOS mirror list by default. Follow instructions in that file if changes are needed.

You can point Yum 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.

For applications that need to be installed inside Guest Shell, go to the CentOS 7 repo at http://mirror.centos.org/centos/7/os/x86_64/Packages/.

Yum resolves the dependencies and installs all the required packages.

```
[guestshell@guestshell ~]$ sudo chvrf management yum -y install glibc.i686
```

```
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: bay.uchicago.edu
* extras: pubmirrors.dal.coreospace.com
* updates: mirrors.cmich.edu
Resolving Dependencies
"-->" Running transaction check
"--->" Package glibc.i686 0:2.17-78.el7 will be installed
"-->" Processing Dependency: libfreebl3.so(NSSRAWHASH_3.12.3) for package:
glibc-2.17-78.el7.i686
"-->" Processing Dependency: libfreebl3.so for package: glibc-2.17-78.el7.i686
"-->" Running transaction check
"--->" Package nss-softokn-freebl.i686 0:3.16.2.3-9.el7 will be installed
"-->" Finished Dependency Resolution
```

Dependencies Resolved

```
Package Arch Version Repository Size
```

```
Installing:
glibc i686 2.17-78.el7 base 4.2 M
Installing for dependencies:
nss-softokn-freebl i686 3.16.2.3-9.el7 base 187 k
```

Transaction Summary

```
Install 1 Package (+1 Dependent package)
```

```
Total download size: 4.4 M
Installed size: 15 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
(1/2): nss-softokn-freebl-3.16.2.3-9.el7.i686.rpm | 187 kB 00:00:25
(2/2): glibc-2.17-78.el7.i686.rpm | 4.2 MB 00:00:30
```

```
Total 145 kB/s | 4.4 MB 00:00:30
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : nss-softokn-freebl-3.16.2.3-9.el7.i686 1/2
Installing : glibc-2.17-78.el7.i686 2/2
error: lua script failed: [string "%triggerin(glibc-common-2.17-78.el7.x86_64)":1: attempt
to compare number with nil
Non-fatal "<"unknown">" scriptlet failure in rpm package glibc-2.17-78.el7.i686
Verifying : glibc-2.17-78.el7.i686 1/2
Verifying : nss-softokn-freebl-3.16.2.3-9.el7.i686 2/2
```

```
Installed:
glibc.i686 0:2.17-78.e17

Dependency Installed:
nss-softokn-freebl.i686 0:3.16.2.3-9.e17

Complete!
```



Note When more space is needed in the Guest Shell root file system for installing or running packages, use the **guestshell resize roots *size-in-MB*** command to increase the file system size.



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

Security Posture for Guest Shell

Use of the Guest Shell in Cisco Nexus 3400-S series devices is just one of the many ways the network Admin can manage or extend the functionality of the system. The Guest Shell is intended to provide 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.

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 3400-S 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 uses 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.

Namespace Isolation

The Guest Shell environment runs within a Linux container that uses various namespaces to decouple the Guest Shell execution space from that of the host. The Guest Shell is run in a separate user namespace, which helps protect the integrity of the host system, as processes running as root within the Guest Shell are not root of the host. These processes appear to be running as UID 0 within the Guest Shell due to UID mapping, but the kernel knows the real UID of these processes and evaluates the POSIX capabilities within the appropriate user namespace.

When a user enters the Guest Shell from the host, a user of the same name is created within the Guest Shell. While the names match, the UID of the user within the Guest Shell is not the same as the UID on the host. To still allow users within the Guest Shell to access files on shared media (for example, `/bootflash` or `/volatile`), the common NX-OS GID that are used on the host (for example, `network-admin` or `network-operator`) are mapped into the Guest Shell such that the values are the same and the Guest Shell instance of the user is associated with the appropriate groups based on group membership on the host.

As an example, consider user `bob`. On the host, `bob` has the following UID and GID membership:

```
bash-4.3$ id
uid=2004(bob) gid=503(network-admin) groups=503(network-admin),504(network-operator)
```

When user `bob` is in the Guest Shell, the group membership from the host is set up in the Guest Shell:

```
[bob@guestshell ~]$ id
uid=1002(bob) gid=503(network-admin)
groups=503(network-admin),504(network-operator),10(wheel)
```

Files that are created by user `bob` in the host Bash shell and the Guest Shell have different owner IDs. The example output below shows that the file created from within the Guest Shell has owner ID 12002, instead of 1002 as shown in the example output above. This is due to the command being issued from the host Bash shell and the ID space for the Guest Shell starting at ID 11000. The group ID of the file is `network-admin`, which is 503 in both environments.

```
bash-4.3$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 12002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 2004 503 4 Jun 22 15:47 /bootflash/bob_host
```

```
bash-4.3$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 12002 network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

The user is allowed to access the file due to the file permission settings for the `network-admin` group, and the fact that `bob` is a member of `network-admin` in both the host and Guest Shell.

Inside the Guest Shell environment, the example output below shows that the owner ID for the file that is created by `bob` from the host is 65534. This indicates that the actual ID is in a range that is outside the range of IDs that are mapped into the user namespace. Any unmapped ID will be shown as this value.

```
[bob@guestshell ~]$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 1002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 65534 503 4 Jun 22 15:47 /bootflash/bob_host
```

```
[bob@guestshell ~]$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 65534 network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

Root-User Restrictions

As a best practice for developing secure code, Cisco recommends 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 the Guest Shell 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 the Guest Shell follow:

- `cap_audit_control`
- `cap_audit_write`
- `cap_mac_admin`
- `cap_mac_override`
- `cap_mknod`
- `cap_net_broadcast`
- `cap_sys_boot`
- `cap_syslog`
- `cap_sys_module`
- `cap_sys_nice`
- `cap_sys_pacct`
- `cap_sys_ptrace`
- `cap_sys_rawio`
- `cap_sys_resource`
- `cap_sys_time`
- `cap_wake_alarm`

While the `net_admin` capability is not dropped, user namespace and the host ownership of the network namespaces prevents the Guest Shell user from modifying the interface state. As root within the Guest Shell, bind mounts may be used as well as tmpfs and ramfs mounts. Other mounts are prevented.

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 between Guest Shell and services on the host.

Guest File System Access Restrictions

To preserve the integrity of the files within the Guest Shell, the file systems of the Guest Shell are not accessible from the NX-OS CLI.

`bootflash:` and `volatile:` of the host are mounted as `/bootflash` and `/volatile` within the Guest Shell. A network-admin can access files on this media using the NX-OS `exec` commands from the host or using Linux commands from within the Guest Shell.

Managing the Guest Shell

The following are commands to manage the Guest Shell:

Table 2: Guest Shell CLI Commands

Commands	Description
<code>guestshell enable</code> { <code>package</code> [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}	<ul style="list-style-type: none"> When <i>guest shell OVA file</i> is specified: 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. When <i>rootfs-file-URI</i> is specified: Imports a Guest Shell rootfs when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.
<code>guestshell export rootfs package</code> <i>destination-file-URI</i>	Exports a Guest Shell rootfs file to a local URI (bootflash, USB1, and so on).
<code>guestshell disable</code>	Shuts down and disables the Guest Shell.

Commands	Description
<p>guestshell upgrade {package [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}</p>	<ul style="list-style-type: none"> When <i>guest shell OVA file</i> is specified: <p>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.</p> <p>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.</p> <p>You are prompted for a confirmation before carrying out the upgrade command.</p> When <i>rootfs-file-URI</i> is specified: <p>Imports a Guest Shell rootfs file when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.</p>
<p>guestshell reboot</p>	<p>Deactivates the Guest Shell and then reactivates it. You are prompted for a confirmation before carrying out the reboot command.</p> <p>Note This is the equivalent of a guestshell disable command followed by a guestshell enable command in EXEC mode.</p> <p>This is useful when processes inside the Guest Shell have been stopped and need to be restarted. The run guestshell command relies on <code>sshd</code> running in the Guest Shell.</p> <p>If the command does not work, the <code>sshd</code> 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.</p>

Commands	Description
guestshell destroy	<p>Deactivates and uninstalls the Guest Shell. All resources that are associated with the Guest Shell are returned to the system. The show virtual-service global command indicates when these resources become available.</p> <p>Issuing this command results in a prompt for a confirmation before carrying out the destroy command.</p>
guestshell run guestshell	Connects to the Guest Shell that is already running with a shell prompt. No username or password is required.
guestshell run <i>command</i> run guestshell <i>command</i>	<p>Executes a Linux or UNIX command within the context of the Guest Shell environment.</p> <p>After execution of the command you are returned to the switch prompt.</p>
guestshell resize [cpu memory rootfs]	<p>Changes the allotted resources available for the Guest Shell. The changes take effect the next time the Guest Shell is enabled or rebooted.</p> <p>Note Resize values are cleared when the guestshell destroy command is used.</p>
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 the Standby supervisor to use the same rootfs 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	<p>If the guest shell 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 extra virtual-services can be installed or enabled after issuing this command until after the system has been reloaded.</p> <p>You are prompted for a confirmation before initiating the reset.</p>



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, LXC's 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           guestshell.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 you destroy the Guest Shell and reload the system, 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 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 is 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 that is 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
```

Enabling the Guest Shell in Base Boot Mode

You can choose to boot your system in *base boot mode*. When you boot your system in base boot mode, the Guest Shell is not started by default. In order to use the Guest Shell in this mode, you must activate the RPMs containing the virtualization infrastructure as well as the Guest Shell image. After, the Guest Shell and virtual-service commands are available.

If the RPM activation commands are run in this order:

1. `install activate guestshell`
2. `install activate virtualization`

The Guest Shell container will be activated automatically as it would have been if the system had been booted in full mode.

If the RPM activation commands are run in the reverse order:

1. `install activate virtualization`
2. `install activate guestshell`

Then the Guest Shell will not be enabled until you run the **guestshell enable** command.

Replicating the Guest Shell

The Cisco Nexus 3400-S has a Guest Shell **rootfs** which can be customized on one switch and can be deployed onto multiple switches.

The approach is to customize and then export the Guest Shell **rootfs** and store it on a file server. A POAP script can download (import) the Guest Shell **rootfs** to other switches and install the specific Guest Shell across many devices simultaneously.

Exporting Guest Shell **rootfs**

Use the **guestshell export rootfs package *destination-file-URI*** command to export a Guest Shell **rootfs**.

The *destination-file-URI* parameter is the name of the file that the Guest Shell **rootfs** is copied to. This file allows for local URI options (bootflash, USB1, and so on).

The **guestshell export rootfs package** command:

- Disables the Guest Shell (if already enabled).
- Creates a guest shell import YAML file and inserts it into the /cisco directory of the **rootfs** ext4 file.
- Copies the **rootfs** ext4 file to the target URI location.
- Re-enables the Guest Shell if it had been previously enabled.

Importing Guest Shell **rootfs**

When importing a Guest Shell **rootfs**, there are two situations to consider:

- Use the **guestshell enable package *rootfs-file-URI*** command to import a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.

- Use the **guestshell upgrade package** *rootfs-file-URI* command to import a Guest Shell **rootfs** when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.

The *rootfs-file-URI* parameter is the **rootfs** file that is stored on local storage (bootflash, USB, and so on).

When this command is executed with a file that is on bootflash, the file is moved to a storage pool on bootflash.

As a best practice, you should copy the file to the bootflash and validate the md5sum before using the **guestshell upgrade package** *rootfs-file-URI* command.



Note The **guestshell upgrade package** *rootfs-file-URI* command can be executed from within the Guest Shell.



Note The rootfs file is not a Cisco signed package, you must configure to allow unsigned packages before enabling as shown in the example.

```
(config-virt-serv-global)# signing level unsigned
Note: Support for unsigned packages has been user-enabled. Unsigned packages are not endorsed
by Cisco. User assumes all responsibility.
```



Note To restore the embedded version of the rootfs:

- Use the **guestshell upgrade** command (without extra parameters) when the Guest Shell has already been installed.
- Use the **guestshell enable** command (without extra parameters) when the Guest Shell had been destroyed.



Note When running this command from within a Guest Shell, or outside a switch using NX-API, you must set **terminal dont-ask** to skip any prompts.

The **guestshell enable package** *rootfs-file-URI* command:

- Performs basic validation of the **rootfs** file.
- Moves the **rootfs** into the storage pool.
- Mounts the **rootfs** to extract the YAML file from the /cisco directory.
- Parses the YAML file to obtain VM definition (including resource requirements).
- Activates the Guest Shell.

Example workflow for **guestshell enable** :

```
switch# copy scp://user@10.1.1.1/my_storage/gs_rootfs.ext4 bootflash: vrf management
switch# guestshell resize cpu 8
```

```
Note: System CPU share will be resized on Guest shell enable
switch# guestshell enable package bootflash:gs_rootfs.ext4
Validating the provided rootfs
switch# 2017 Jul 31 14:58:01 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual
service 'guestshell+'
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
'guestshell+'
2017 Jul 31 14:58:33 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```



Note Workflow for **guestshell upgrade** is preceded by the existing Guest Shell being destroyed.



Note Resize values are cleared when the **guestshell upgrade** command is used.

Importing YAML File

A YAML file that defines some user modifiable characteristics of the Guest Shell is automatically created as a part of the export operation. The YAML file is embedded into the Guest Shell **rootfs** in the /cisco directory. The YAML file is not a complete descriptor for the Guest Shell container. The file only contains some of the user-modifiable parameters.

Example of a guest shell import YAML file:

```
---
import-schema-version: "1.0"
info:
  name: "GuestShell"
  version: "2.4.0"
  description: "Exported GuestShell: 20170216T175137Z"
app:
  apptype: "lxc"
  cpuarch: "x86_64"
  resources:
    cpu: 3
    memory: 307200
    disk:
      - target-dir: "/"
        capacity: 250
  ...
```

The YAML file is generated when the **guestshell export rootfs package** command is executed. The file captures the values of the currently running Guest Shell.

The info section contains non-operational data that is used to help identify the Guest Shell. Some of the information will be displayed in the output of the **show guestshell detail** command.

The description value is an encoding of the UTC time when the YAML file was created. The time string format is the same as DTSTAMP in RFC5545 (iCal).

The resources section describes the resources that are required for hosting the Guest Shell. The value "/" for the target-dir in the example identifies the disk as the **rootfs**.



Note If you specified resized values while the Guest Shell was destroyed, those values take precedence over the values in the import YAML file when you use the **guestshell enable package** command.

The `cpuarch` value indicates the CPU architecture that is expected for the container to run.

You can modify the YAML file (such as the description or increase the resource parameters, if appropriate) after the export operation is complete.

Cisco provides a Python script that you can run to validate a modified YAML file with a JSON schema. It is not meant to be a complete test (for example, device-specific resource limits are not checked), but it is able to flag common errors. The Python script with examples is located at https://github.com/datacenter/opennxos/tree/master/guestshell_import_export. The following JSON file describes the schema for version 1.0 of the Guest Shell import YAML.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Guest Shell import schema",
  "description": "Schema for Guest Shell import descriptor file - ver 1.0",
  "copyright": "2017 by Cisco systems, Inc. All rights reserved.",
  "id": "",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "import-schema-version": {
      "id": "/import-schema-version",
      "type": "string",
      "minLength": 1,
      "maxLength": 20,
      "enum": [
        "1.0"
      ]
    },
    "info": {
      "id": "/info",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "id": "/info/name",
          "type": "string",
          "minLength": 1,
          "maxLength": 29
        },
        "description": {
          "id": "/info/description",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "version": {
          "id": "/info/version",
          "type": "string",
          "minLength": 1,
          "maxLength": 63
        },
        "author-name": {
          "id": "/info/author-name",
          "type": "string",
          "minLength": 1,

```



```

        "maxLength": 199
    },
    "author-link": {
        "id": "/info/author-link",
        "type": "string",
        "minLength": 1,
        "maxLength": 199
    }
}
},
"app": {
    "id": "/app",
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "apptype": {
            "id": "/app/apptype",
            "type": "string",
            "minLength": 1,
            "maxLength": 63,
            "enum": [
                "lxc"
            ]
        },
        "cpuarch": {
            "id": "/app/cpuarch",
            "type": "string",
            "minLength": 1,
            "maxLength": 63,
            "enum": [
                "x86_64"
            ]
        },
        "resources": {
            "id": "/app/resources",
            "type": "object",
            "additionalProperties": false,
            "properties": {
                "cpu": {
                    "id": "/app/resources/cpu",
                    "type": "integer",
                    "multipleOf": 1,
                    "maximum": 100,
                    "minimum": 1
                },
                "memory": {
                    "id": "/app/resources/memory",
                    "type": "integer",
                    "multipleOf": 1024,
                    "minimum": 1024
                },
                "disk": {
                    "id": "/app/resources/disk",
                    "type": "array",
                    "minItems": 1,
                    "maxItems": 1,
                    "uniqueItems": true,
                    "items": {
                        "id": "/app/resources/disk/0",
                        "type": "object",
                        "additionalProperties": false,
                        "properties": {
                            "target-dir": {
                                "id": "/app/resources/disk/0/target-dir",

```

```

        "type": "string",
        "minLength": 1,
        "maxLength": 1,
        "enum": [
            "/"
        ]
    },
    "file": {
        "id": "/app/resources/disk/0/file",
        "type": "string",
        "minLength": 1,
        "maxLength": 63
    },
    "capacity": {
        "id": "/app/resources/disk/0/capacity",
        "type": "integer",
        "multipleOf": 1,
        "minimum": 1
    }
}
}
},
"required": [
    "memory",
    "disk"
]
}
},
"required": [
    "apptype",
    "cpuarch",
    "resources"
]
}
},
"required": [
    "app"
]
}
}

```

show guestshell Command

The output of the **show guestshell detail** command includes information that indicates whether the Guest Shell was imported or was installed from an OVA.

Example of the **show guestshell detail** command after importing **rootfs**.

```

switch# show guestshell detail
Virtual service guestshell+ detail
State                : Activated
Package information
Name                 : guestshell.ova
Path                 : /isanboot/bin/guestshell.ova
Application
Name                 : GuestShell
Installed version    : 2.4(0.0)
Description          : Cisco Systems Guest Shell
Signing
Key type             : Cisco release key
Method               : SHA-1
Licensing
Name                 : None
Version              : None

```

```

Resource reservation
Disk                : 190 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

```

Verifying Virtual Service and Guest Shell Information

You can verify virtual service and Guest Shell information with the following commands:

Command	Description
<pre> show virtual-service global switch# show virtual-service global Virtual Service Global State and Virtualization Limits: Infrastructure version : 1.10 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) 5376 256 5120 bootflash (MB) 8192 190 8002 switch# </pre>	<p>Displays the global state and limits for virtual services.</p>
<pre> show virtual-service list switch# show virtual-service list Virtual Service List: Name Status Package Name ----- guestshell+ Activated guestshell.ova </pre>	<p>Displays a summary of the virtual services, the status of the virtual services, and installed software packages.</p>

Command	Description
<pre> show guestshell detail switch# show guestshell detail Virtual service guestshell+ detail State : Activated Package information Name : guestshell.ova Path : /isanboot/bin/guestshell.ova Application Name : GuestShell Installed version : 2.4(0.0) Description : Cisco Systems Guest Shell Signing Key type : Cisco release key Method : SHA-1 Licensing Name : None Version : None Resource reservation Disk : 190 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 </pre>	<p>Displays details about the guest shell package (such as version, signing resources, and devices).</p>

Persistently Starting Your Application from the Guest Shell

Your application should have a `systemd / systemctl` service file that gets installed in `/usr/lib/systemd/system/application_name.service`. This service file should have the following general format:

```

[Unit]
Description=Put a short description of your application here

[Service]
ExecStart=Put the command to start your application here
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target

```



Note To run `systemd` as a specific user, add `User=<username>` to the `[Service]` section of your service.

Procedure for Persistently Starting Your Application from the Guest Shell

Procedure

-
- Step 1** Install your application service file that you created above into `/usr/lib/systemd/system/application_name.service`
 - Step 2** Start your application with `systemctl start application_name`
 - Step 3** Verify that your application is running with `systemctl status -l application_name`
 - Step 4** Enable your application to be restarted on reload with `systemctl enable application_name`
 - Step 5** Verify that your application is running with `systemctl status -l application_name`
-

An Example Application in the Guest Shell

The following example demonstrates an application in the Guest Shell:

```
root@guestshell guestshell]# cat /etc/init.d/hello.sh
#!/bin/bash

OUTPUTFILE=/tmp/hello

rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
[root@guestshell guestshell]#
[root@guestshell guestshell]#
[root@guestshell system]# cat /usr/lib/systemd/system/hello.service
[Unit]
Description=Trivial "hello world" example daemon

[Service]
ExecStart=/etc/init.d/hello.sh &
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
[root@guestshell system]#
[root@guestshell system]# systemctl start hello
[root@guestshell system]# systemctl enable hello
[root@guestshell system]# systemctl status -l hello
hello.service - Trivial "hello world" example daemon
   Loaded: loaded (/usr/lib/systemd/system/hello.service; enabled)
   Active: active (running) since Sun 2015-09-27 18:31:51 UTC; 10s ago
   Main PID: 355 (hello.sh)
   CGroup: /system.slice/hello.service
           ##355 /bin/bash /etc/init.d/hello.sh &
           ##367 sleep 10

Sep 27 18:31:51 guestshell hello.sh[355]: Executing: /etc/init.d/hello.sh &
[root@guestshell system]#
```

```
[root@guestshell guestshell]# exit
exit
[guestshell@guestshell ~]$ exit
logout
switch# reload
This command will reboot the system. (y/n)? [n] y
```

After reload

```
[root@guestshell guestshell]# ps -ef | grep hello
root      20      1  0 18:37 ?          00:00:00 /bin/bash /etc/init.d/hello.sh &
root      123     108  0 18:38 pts/4      00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# cat /tmp/hello
Sun Sep 27 18:38:03 UTC 2015
Hello World
Sun Sep 27 18:38:13 UTC 2015
Hello World
Sun Sep 27 18:38:23 UTC 2015
Hello World
Sun Sep 27 18:38:33 UTC 2015
Hello World
Sun Sep 27 18:38:43 UTC 2015
Hello World
[root@guestshell guestshell]#
```

Running under `systemd` / `systemctl`, your application is automatically restarted if it dies (or if you kill it). The Process ID is originally 226. After killing the application, it is automatically restarted with a Process ID of 257.

```
[root@guestshell guestshell]# ps -ef | grep hello
root      226      1  0 19:02 ?          00:00:00 /bin/bash /etc/init.d/hello.sh &
root      254     116  0 19:03 pts/4      00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# kill -9 226
[root@guestshell guestshell]#
[root@guestshell guestshell]# ps -ef | grep hello
root      257      1  0 19:03 ?          00:00:00 /bin/bash /etc/init.d/hello.sh &
root      264     116  0 19:03 pts/4      00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
```

Troubleshooting Guest Shell Issues

Unable to Access Files on Bootflash from Root in the Guest Shell

You may find that you are unable to access files on the bootflash from root in the Guest Shell.

From the host:

```
root@switch# ls -al /bootflash/try.that
-rw-r--r-- 1 root root 0 Apr 27 20:55 /bootflash/try.that
root@switch#
```

From the Guest Shell:

```
[root@guestshellbootflash]# ls -al /bootflash/try.that
-rw-r--r-- 1 65534 host-root 0 Apr 27 20:55 /bootflash/try.that
[root@guestshellbootflash]# echo "some text" >> /bootflash/try.that
-bash: /bootflash/try.that: Permission denied
[root@guestshellbootflash]#
```

This might be because the user namespace is being used to protect the host system, so the root in the Guest Shell is not actually the root of the system.

To recover from this issue, verify that the file permissions and group-id of the files allow for shared files on bootflash to be accessed as expected. You may need to change the permissions or group-id from the host Bash session.



CHAPTER 5

Innovium Shell

This chapter contains the following topics:

- [About the Innovium Shell, on page 65](#)
- [Guidelines and Limitations, on page 65](#)
- [Accessing the Innovium Shell Through CLI API, on page 66](#)
- [Passing Commands to the Innovium Shell, on page 68](#)

About the Innovium Shell

The Cisco Nexus 3400-S series of switches consists of the following:

- Cisco Nexus 3408-S, which is a 4-RU switch with 8 module slots. The module slots can accept a mix of line-card expansion modules (LEMs) for extra high-speed connectivity.
- Cisco Nexus 3432D-S, which is a 1-RU chassis switch with a fixed port configuration. This switch has no module slots or LEMs.

Cisco Nexus 3400-S switches use Innovium Teralynx ASICs, which exist on the switch motherboard. A specific shell enables access directly to the ASICs, or passing ASIC commands to the shell from Cisco NX-OS. This shell is the Innovium shell (innoshell).

Guidelines and Limitations

Following are the guidelines and limitations for the Innovium shell (innoshell):

- The innoshell is for authorized use only. Use it with extreme caution and only when authorized by Cisco.
- You can access and read information from the Innovium Teralynx ASICs without any limitations. However, Cisco does not recommend changing the Innovium configuration settings.
- The Innovium shell is supported on the Cisco Nexus 3408-S and the Cisco Nexus 3432D-S switches only.

Accessing the Innovium Shell Through CLI API

The Innovium shell is available directly from EXEC mode at the top level of the NX-OS command hierarchy. The Innovium shell enables you to issue commands directly to the Innovium Teralynx ASICs.

```
switch-1# ?
...
gunzip          Uncompresses LZ77 coded files
gzip           Compresses file using LZ77 coding
hardware       Change hardware usage settings
innoshell     Innoshell
install        Upgrade software
ip             Display IP information
key            Encryption key for strong encryption
...
```

As an alternative, you can enter individual Innovium-shell commands directly from NX-OS. See [Passing Commands to the Innovium Shell](#), on page 68.

The Innovium shell is supported on the Innovium Teralynx ASICs, which are in Cisco Nexus 3408-S and Cisco Nexus 3432D-S switches. You can check the model of your switch by entering the **show module** command.

Example:

```
switch-1# show module
Mod Ports Module-Type                               Model                               Status
-----
1      0      Supervisor                                       N3K-C3408-S          active *
```

This topic documents how to access the Innovium shell from NX-OS, use the online Help function, and exit the Innovium shell. For detailed documentation about the Innovium shell and its commands, consult the Innovium Teralynx documentation.

- To enter the Innovium shell, enter the **innoshell** command.

Example:

```
switch-1# innoshell
Warning: innoshell access should be used with caution
exec /lc/isan/bin/innoshell
moving ifcsrshell.log to ifcsrshell.log.bkp
Script started, file is ifcsrshell.log
          Connected to Innovium Shell Server

          Innovium Remote Command Shell Client.
          Type '?' or 'help' for help. Type 'exit' or 'quit' to exit shell.
IVM-R:0>
```

The command prompt changes to `IVM-R:0` to indicate that you are in the Innovium shell.

- To get a list of top-level commands in the hierarchy, type **?** (question mark) or **help**.

Example:

```
IVM-R:0>?
+-----+
| Innovium CLI Shell Help Menu : Description |
|                                           |
+-----+
```

```

| exit                               : Quit the Innovium shell
| source                             : Source a command file with shell commands
| redirect                           : Redirect outputs from a shell to a file
| pen                                 : Pen Access Commands
| ifcs                               : IFCS Api Debug Commands
| console                             : Start an interactive Python Interpreter from within
the shell |
| clear                               : Clear screen
| run                                 : Clear screen
-----+-----

```

- To get detailed syntax help for a command, type the command name plus ? (question mark) or **help**.

Example:

```
IVM-R:0>pen ?
```

```

Usage:
  pen read ib <ib#> pen_name <index>      - node pen read <pen_id>
...
  pen write ib <ib#> pen_name <index>     - node pen write <pen_id>
...
  pen modify ib <ib#> pen_name <index>    - node pen modify <pen_id>
...
  pen insert ib <ib#> pen_name <index>    - node pen insert <pen_id>
...
  pen lookup ib <ib#> pen_name <index>    - node pen lookup <pen_id>
...
  pen delete ib <ib#> pen_name <index>    - node pen delete <pen_id>
...
  pen flush ib <ib#> <pen_id>            - node pen flush <pen_id>
...

  pen help or ?                          - show this text
Command Details:
  Type : "pen <subcmd> ? or pen <subcmd> help" For more specific sub
cmd help

```

- Notice that command help is nested, so you can get help about subcommands.

Example:

```
IVM-R:0>pen read ib ?
```

```

Usage::
  pen read ib <ib> [pic <pic_id>] <pen-name> [count] - node pen read
pens)
Command Options:
  <ib>                               - IB #
  [pic-id]                             - pic_id (mandatory for pic
pens)
  <pen_name>                           - pen name
  [count]                               - number of entries (optional)
  <index>                               - Read index (mandatory for
Direct index pen)

```

```
IVM-R:0>
```

- To exit the Innovium shell, enter either the **exit** or **quit** command, which terminates the current Innovium CLI session and returns you to the NX-OS prompt.

Example:

```
IVM-R:0>exit
Script done, file is ifcsrshell.log
switch-1#
```

Passing Commands to the Innovium Shell

Besides logging in to the Innovium shell, you can pass Innovium shell commands directly from NX-OS to the Innovium shell. The command is executed in the Innovium shell, and then returns its output to NX-OS. Passing Innovium commands from Cisco NX-OS to the Innovium shell is helpful if you have the exact syntax of the command, or if you have just a few tasks to do in the Innovium shell.

To pass the command to the Innovium shell, issue the **innoshell cmd** command and enclose the exact command syntax in double quotes.

Example: This example is equivalent to:

1. Logging in to the Innovium shell
2. Issuing **ifcs show ecmp**
3. Logging out of the Innovium shell

```
switch-1# innoshell cmd "ifcs show ecmp"
exec /lc/isan/bin/innoshell ifcs show ecmp
Script started on Tue Apr 16 13:47:49 2019
ifcs show ecmp
exit

Connected to Innovium Shell Server

Innovium Remote Command Shell Client.
Type '?' or 'help' for help. Type 'exit' or 'quit' to exit shell.
IVM-R:0>ifcs show ecmp
Total ecmp count: 3
+-----+
|      ecmp |   type | number_of_slots | group_size | member_count |
+-----+
| (ecmp:  1) | DEFAULT |          0 |          0 |          2 |
| (ecmp:  2) | DEFAULT |          0 |          0 |          2 |
| (ecmp:  4) | DEFAULT |          0 |          0 |          4 |
+-----+

IVM-R:0>exit

Script done on Tue Apr 16 13:47:55 2019

switch-1#
```



CHAPTER 6

Python API

This chapter contains the following topics:

- [About the Python API, on page 69](#)
- [Using Python, on page 69](#)

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 more documentation.

The Cisco Nexus 3400-S Series devices support Python v2.7.5 and later in both interactive and noninteractive (script) modes and are available in the Guest Shell.

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

For more Python modules, you can install the `python-modules-nxos` RPM (`python-modules-nxos-1.0.0-XXX.lib32_x86.rpm`, where `XXX` is the NX-OS version) from <https://devhub.cisco.com/artifactory/open-nxos/>. Refer to the "Manage Feature RPMs" section for instructions about installing an RPM.

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 must 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 3: CLI Command APIs

API	Description
cli() Example: <pre>string = cli ("cli-command")</pre>	Returns the raw output of CLI commands, including control or special characters. Note The interactive Python interpreter prints control or special characters 'escaped'. A carriage return prints as '\n' and gives results that can be difficult to read. The clip() API gives results that are more readable.
clid() Example: <pre>json_string = clid ("cli-command")</pre>	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: <pre>clip ("cli-command")</pre>	Prints the output of the CLI command directly to stdout and returns nothing to Python. Note <pre>clip ("cli-command")</pre> Is equivalent to <pre>r=cli("cli-command") print r</pre>

When you run two or more commands 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 you run two or more commands 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. You must surround the semicolon (;) 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 2.7.11 (default, Feb 26 2018, 03:34:16)
[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(cli('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(cli('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                                kern_uptm_secs = 21
                                kick_file_name = bootflash:///nxos.9.2.2.bin.S246
                                    rr_service = None
                                        module_id = Supervisor Module
                                            kick_tmstamp = 07/11/2018 00:01:44
                                                bios_cmpl_time = 05/17/2018
                                                    bootflash_size = 20971520
                                                        kickstart_ver_str = 9.2(2t)
                                                            kick_cmpl_time = 5/6/2019 9:00:00
                                                                chassis_id = Nexus3000 3408-S (8 Slot) Chassis
                                                                    proc_board_id = SAL171211LX
                                                                        memory = 16077872
                                                                            manufacturer = Cisco Systems, Inc.
                                                                                kern_uptm_mins = 26
                                                                                    bios_ver_str = 05.31
                                                                                        cpu_name = Intel(R) Xeon(R) CPU D-1528 @ 1.90GHz
                                                                                            kern_uptm_hrs = 2
                                                                                                rr_usecs = 816550
                                                                                                    rr_sys_ver = 9.2(2t)
                                                                                                        rr_reason = Reset Requested by CLI command reload
                                                                                                            rr_ctime = Wed May 8 20:44:39 2019
                                                                                                                header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2019, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.
                                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 3400-S 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' % \
        (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 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 3400-S 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
!Running configuration last done at: Mon Nov 25 09:32:27 2019
!Time: Mon Nov 25 10:06:04 2019

version 9.2(2t) Bios:version 05.35
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 that is 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 3400-S switches, 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 in-band interface by switching to a desired virtual routing context.

```
switch# python
Python 2.7.11 (default, Feb 26 2018, 03:34:16)
[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 who are 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 nonprivileged users. Nonprivileged users have limited access to Cisco NX-OS resources, such as the 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 2.7.11 (default, Feb 26 2018, 03:34:16)
[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 nonprivileged user being denied access:

```
switch# python
Python 2.7.11 (default, Feb 26 2018, 03:34:16)
[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 nonprivileged 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 7

Scripting with TCL

This chapter contains the following topics:

- [About Tcl, on page 79](#)
- [Running the Tcsh Command, on page 82](#)
- [Navigating Cisco NX-OS Modes from the Tcsh Command, on page 82](#)
- [Tcl References, on page 83](#)

About Tcl

Tcl (pronounced "tickle") is a scripting language that increases flexibility of CLI commands. 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.

Guidelines and Limitations

Following are guidelines and limitations for TCL scripting:

- The Cisco Nexus 3400-S switch supports TCL.
- Some processes and **show** commands can cause a large amount of output. If you are running scripts, and need to terminate long-running output, use Ctrl+C (not Ctrl+Z) to terminate the command output. If you use Ctrl+Z, a SIGCONT (signal continuation) message can be generated, which can cause the script to halt. Scripts that are halted through SIGCONT messages require your intervention to resume operation.

Tcsh 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 preceding 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 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 fails and the Tcl shell displays 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 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 terminates 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 initial 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 iterations.

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.

Procedure

	Command or Action	Purpose
Step 1	<pre>tclsh [bootflash:filename [argument ...]]</pre> <p>Example:</p> <pre>switch# tclsh ? <CR> bootflash: The file to run</pre>	<p>Starts a Tcl shell.</p> <p>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.</p> <p>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.</p>

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.

Procedure

	Command or Action	Purpose
Step 1	<pre>tclsh</pre> <p>Example:</p> <pre>switch# tclsh switch-tcl#</pre>	Starts an interactive Tcl shell.
Step 2	<pre>configure terminal</pre> <p>Example:</p> <pre>switch-tcl# configure terminal switch(config-tcl)#</pre>	<p>Runs a Cisco NX-OS command in the Tcl shell, changing modes.</p> <p>Note The Tcl prompt changes to indicate the Cisco NX-OS command mode.</p>

	Command or Action	Purpose
Step 3	tclquit Example: <pre>switch-tcl# tclquit switch#</pre>	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.



CHAPTER 8

Kernel Stack

This chapter contains the following topics:

- [About Kernel Stack, on page 85](#)
- [Guidelines and Limitations, on page 85](#)
- [Changing the Port Range, on page 86](#)
- [Netdevice Property Changes, on page 87](#)

About Kernel Stack

Kernel Stack (kstack) uses well known Linux APIs to manage the routes and front panel ports.

Open Containers, like the Guest Shell, are Linux environments that are decoupled from the host software. You can install or modify software within that environment without impacting the host software packages.

Guidelines and Limitations

- Guest shell and the host Bash Shell use Kernel Stack (kstack).
- The Guest Shell and the host Bash Shell start in the default network namespace.
 - Use the **setns** system call to access other network namespaces
 - The **nsenter** and **ip netns exec** utilities can be used to execute within the context of a different network namespace.
- The interface state may be read from `/proc/net/dev` or retrieved using other typical Linux utilities such as **ip**, **ifconfig**, or **netstat**. The counters are for packets that have initiated or terminated on the switch.
- Use **ethtool -S** to get extended statistics from the net devices, which include packets that are switched through the interface.
- You can run packet capture applications like **tcpdump** to capture packets that initiate or terminate on the switch.
- There is no support for networking state changes (interface creation or deletion, IP address configuration, MTU change, and so on) from the Guest Shell.

- IPv4 and IPv6 are supported.
- Raw PF_PACKET is supported.
- Only on stack (Netstack or kstack) at a time can use well-known ports (0-15000), regardless of the network namespace.
- There is no IP connectivity between applications using Nestack and applications running kstack on the same switch. This limitation holds true regardless of whether the kstack applications are being run from the host Bash Shell or within a container.
- Applications within the Guest Shell are not allowed to send packets directly over an Ethernet out-of-band channel (EOBC) interface to communicate with the line cards or standby Sup.
- The management interface (mgmt0) is represented as eth1 in the kernel netdevices.

Changing the Port Range

Netstack and kstack divide the port range between them. The default port ranges are as follows:

- Kstack—From 15001 through 58000
- Netstack—From 58001 through 65535



Note Ports within the range from 63536 through 65535 are reserved for NAT.

Procedure

	Command or Action	Purpose
Step 1	<code>[no] sockets local-port-range start-port end-port</code>	This command modifies the port range for kstack. This command does not modify the Netstack range.

Example

The following example sets the kstack port range:

```
switch# sockets local-port-range 15001 25000
```

What to do next

After you have entered the command, be aware of the following issues:

- Reload the switch after entering the command.
- Leave a minimum of 7000 ports unallocated which are used by Netstack.
- Specify the *start-port* as 15001 or the *end-port* as 65535 to avoid holes in the port range.

Netdevice Property Changes

Netdevices representing the front channel port interfaces are always in the ADMIN UP state. The final, effective state is determined by the link carrier state.

The following example shows the following interfaces in NX-OS, where eth1/17 is shown as **up** and eth1/1 is shown as **down**:

```
root@kstack-switch# sh int ethernet 1/17 brief
Eth1/17      --      eth  routed up      none      1000 (D)  -

root@kstack-switch# sh int ethernet 1/1 brief
Eth1/1      --      eth  routed down    Link not connected      auto (D)  -
```

The following example shows these same interfaces, but this time as shown in the Bash shell using the **ip link show** command:

```
bash-4.3# ip link show Eth1-17
49: Eth1-17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff

bash-4.3# ip link show Eth1-1
33: Eth1-1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff
```

In this example, Eth1-1 is shown as being **UP**, but is shown as **NO-CARRIER** and **state DOWN**.

The following example shows these same interfaces, but this time as shown in the Bash shell using the **ifconfig** command:

```
bash-4.3# ifconfig Eth1-17
Eth1-17  Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:7388 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:100
         RX bytes:0 (0.0 B)  TX bytes:1869164 (1.7 MiB)

bash-4.3# ifconfig Eth1-1
Eth1-1  Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
        inet addr:99.1.1.1  Bcast:99.1.1.255  Mask:255.255.255.0
         UP BROADCAST MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:100
         RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

The output from the **ifconfig** command provides different information, where the **RUNNING** keyword is used to represent the final state. By default, all netdevices show the keyword **UP**, which represents the ADMIN state of the netdevice in the kernel.

The following IPv4 and IPv6 behaviors are applicable to netdevices configured on the Cisco Nexus 3400-S:

- **IPv4 address on netdevices** — The IPv4 addresses are plumbed to the kernel space only when the interface is in the **UP** state. Once plumbed, the IPv4 address continues to stay with the netdevice in the

kernel even if the interface goes **DOWN**. It will be removed only after you have entered the following CLI command to explicitly remove the IP address from the NX-OS interface:

```
Interface Eth1/1
    no ip address IP-address
```

- **IPv6 address on netdevices** — Netdevices are always in the Admin **UP** state, so the IPv6 addresses will not get flushed from the kernel when the interface goes down.



PART II

Applications

- [Cisco Nexus Application Development - SDK, on page 91](#)
- [NX-SDK, on page 99](#)



CHAPTER 9

Cisco Nexus Application Development - SDK

This chapter contains the following topics:

- [About the Cisco SDK, on page 91](#)
- [Installing the SDK, on page 91](#)
- [Procedure for Installation and Environment Initialization, on page 92](#)
- [Using the SDK to Build Applications, on page 93](#)
- [Using RPM to Package an Application, on page 94](#)
- [Creating an RPM Build Environment, on page 95](#)
- [Using General RPM Build Procedure, on page 95](#)
- [Example to Build RPM for collectd with No Optional Plug-Ins, on page 96](#)
- [Example to Build RPM for collectd with Optional Curl Plug-In, on page 97](#)

About the Cisco SDK

The Cisco SDK is a development kit that contains all the tools to build applications for execution on a Cisco Nexus switch running the NX-OS Release 9.2(2t). The basic components are the C cross-compiler, linker, libraries, and header files that are commonly used in many applications. The list is not exhaustive, and you might need to download and build any dependencies that are needed for any particular application. Some applications are ready to be downloaded and used from the Cisco devhub website and do not require building. The SDK can be used to build RPM packages which may be directly installed on a switch.

Installing the SDK

The following lists the system requirements:

- The SDK can run on most modern 64-bit x86_64 Linux systems. It has been verified on CentOS 7 and Ubuntu 14.04. Install and run the SDK under the Bash shell.
- The SDK includes binaries for both 32-bit and 64-bit architectures, so it must be run on an x86_64 Linux system that also has 32-bit libraries installed.

Procedure

Check if the 32-bit libraries are installed:

Example:

```
bash$ ls /lib/ld-linux.so.2
```

If this file exists, then 32-bit libraries should be installed already. Otherwise, install 32-bit libraries as follows:

- For CentOS 7:

```
bash$ sudo yum install glibc.i686
```

- For Ubuntu 14.04:

```
bash$ sudo apt-get install gcc-multilib
```

Procedure for Installation and Environment Initialization

The SDK is available for download at: <https://devhub.cisco.com/artifactory/open-nxos/>.

This file is a self-extracting archive that installs the SDK into a directory of your choice. You are prompted for a path to an SDK installation directory.

```
bash$ ./wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Wind River Linux SDK installer version 8.0-n9000
=====
Enter target directory for SDK (default: /opt/windriver/wrlinux/8.0-n9000):
You are about to install the SDK to "/opt/windriver/wrlinux/8.0-n9000". Proceed[Y/n]? Y
Extracting
SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

Each time that you want to use the SDK in a new shell session, you will need to source the environment setup script. For example:

```
$ . /opt/windriver/wrlinux/8.0-n9000/environment-setup-x86-wrsmlib32-linux
$ . /opt/windriver/wrlinux/8.0-n9000/environment-setup-x86_64-wrs-linux
bash$
```

Use the **source environment-setup-x86-wrsmlib32-linux** and **source environment-setup-x86_64-wrs-linux** commands to add the SDK-specific paths to your shell environment. Add the SDK-specific paths for each shell you intend to use with the SDK. Adding the SDK-specific paths is the key to setting up the SDK to use the correct versions of the build tools and libraries.

Procedure

- Step 1** Browse to the installation directory.

Step 2 Enter the following commands at the Bash prompt:

```
bash$ source environment-setup-x86-wrsmllib32-linux
bash$ source environment-setup-x86_64-wrs-linux
```

Using the SDK to Build Applications

Many of the common Linux build processes work for this scenario. Use the techniques that are best suited for your situation.

The source code for an application package can be retrieved in various ways. For example, you can get the source code either in tar file form or by downloading from a git repository where the package resides.

The following are examples of some of the most common cases.

(Optional) Verify that the application package builds using standard configure/make/make install.

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

Sometimes it is necessary to pass extra options to the `./configure` script, for example to specify which optional components and dependencies are needed. Passing extra options depends entirely on the application being built.

Example - Build Ganglia and its dependencies

In this example, we build ganglia, along with the third-party libraries that it requires - libexpat, libapr, and libconfuse.

libexpat

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
bash$ cd ..
```

libconfuse



Note confuse requires the extra `--enable-shared` option to `./configure`, otherwise it builds a statically linked library instead of the required shared library.

```
bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..
```

ganglia



Note The locations to all the required libraries are passed to `./configure`.

```
bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..
```

Using RPM to Package an Application

If the application successfully builds using "make", then it can be packaged into an RPM.



Note **RPM and spec files**

The RPM package format is designed to package up all files (binaries, libraries, configurations, documents, etc) that are needed for a complete install of the given application. The process of creating an RPM file is therefore somewhat non-trivial. To aid in the RPM build process, a `.spec` file is used that controls everything about the build process.



Note Many third-party applications are available on the internet in the form of source code packaged into tarballs. In many cases, these tarballs will include a .spec file to help with RPM build process. Unfortunately, many of these .spec files are not updated as frequently as the source code itself. Even worse, sometimes there is no spec file at all. In these cases the spec file may need editing or even creating from scratch so that RPMs can be built.

Creating an RPM Build Environment

Before using the SDK to build RPMs, an RPM build directory structure must be created, and some RPM macros set.

Procedure

Step 1 Create the directory structure:

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

Step 2 Set the topdir macro to point to the directory structure created above:

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

Note This step assumes that the current user does not already have a .rpmmacros file that is already set up. If it is inconvenient to alter an existing .rpmmacros file, then the following may be added to all rpmbuild command lines:

```
--define "_topdir ${PWD}"
```

Step 3 Refresh the RPM DB:

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__db.*
bash$ rpm --rebuilddb
```

Note The rpm and rpmbuild tools in the SDK have been modified to use /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm as the RPM database instead of the normal /var/lib/rpm. This modification prevents any conflicts with the RPM database for the host when not using the SDK and removes the need for root access. After SDK installation, the SDK RPM database must be rebuilt through this procedure.

Using General RPM Build Procedure

General RPM Build procedure is as follows:

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app tarball>
bash$ # determine location of spec file in tarball:
```

```
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

The result is a binary RPM in RPMS/ that can be copied to the switch and installed. Installation and configuration of applications can vary. Refer to the application documents for those instructions.

This rpmbuild and installation on the switch is required for every software package that is required to support the application. If a software dependency is required that is not already included in the SDK, the source code must be obtained and the dependencies built. On the build machine, the package can be built manually for verification of dependencies. The following example is the most common procedure:

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

These commands place the build files (binaries, headers, libraries, and so on) into the installation directory. From here, you can use standard compiler and linker flags to pick up the location to these new dependencies. Any runtime code, such as libraries, are required to be installed on the switch also, so packaging required runtime code into an RPM is required.



Note There are many support libraries already in RPM form on the Cisco devhub website.

Example to Build RPM for collectd with No Optional Plug-Ins

Download source tarball and extract spec file:

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

There are four spec files in this tarball. The Red Hat spec file is the most comprehensive and is the only one that contains the correct collectd version. We will use it as an example.

This spec file sets the RPM up to use /sbin/chkconfig to install collectd. However on a Nexus switch, you will use the /usr/sbin/chkconfig instead. Edit the following edited in the spec file:

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```


collectd has numerous optional plug-ins. This spec file enables many plug-ins by default. Many plug-ins have external dependencies, so options to disable these plug-ins must be passed to the **rpmbuild** command line. Instead of typing out one long command line, we can manage the options in a Bash array as follows:

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcached mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>   rpmbuild_opts+=("--without")
>   rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

It is then passed to **rpmbuild** as follows to start the entire build and RPM package process:

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

You can then find the resulting RPMs for *collectd* in the RPMS directory.

These RPM files can now be copied to the switch and installed from the switch Bash shell:

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

Example to Build RPM for collectd with Optional Curl Plug-In

The *collectd curl* plug-in has *libcurl* as a dependency.

In order to satisfy this link dependency during the RPM build process, it is necessary to download and build *curl* under the SDK:

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```



Note The *curl* binaries and libraries are installed to `/path/to/curl-install`. This directory will be created if it does not already exist, so you must have write permissions for the current user. Next, download the source tarball and extract the spec file. This step is exactly the same as in the *collectd* example for no plug-ins.

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
```

```
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```



Note There are four spec files in this tarball. The Red Hat spec file is the most comprehensive, and it is the only one to contain the correct collectd version. We will use that one as an example.

This spec file sets the RPM up to use `/sbin/chkconfig` to install collectd. However on a Cisco Nexus switch, you must use `/usr/sbin/chkconfig` instead, so the following can be edited in the spec file:

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

Here a deviation from the previous example is encountered. The collectd rpmbuild process needs to know the location of libcurl. Edit the collectd spec file to add the following.

Find the string `%configure` in `SPECS/collectd.spec`. This line and those following it define the options that rpmbuild will pass to the `./configure` script.

Add the following option:

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

Next a Bash array is built again to contain the rpmbuild command options. Note the following differences:

- `curl` is removed from the list of plug-ins not to be built
- The addition of `--with curl=force`

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java memcached mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
> rpmbuild_opts+=("--without")
> rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

It is then passed to rpmbuild as follows to start the entire build and RPM package process:

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

The resulting RPMs in the RPMs directory will now also include collectd-curl. These RPM files can now be copied to the switch and installed from the switch Bash shell:

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```



CHAPTER 10

NX-SDK

This chapter contains the following topics:

- [About the NX-SDK, on page 99](#)
- [Install the NX-SDK, on page 100](#)
- [Building and Packaging C++ Applications, on page 101](#)
- [Installing and Running Custom Applications, on page 103](#)

About the NX-SDK

The Cisco NX-OS SDK (NX-SDK) is a C++ abstraction and plug-in library-layer that streamlines access to infrastructure for automation and custom native application creation, such as generating custom:

- CLIs
- Syslogs
- Event and Error managers
- Inter-application communication
- High availability (HA)
- Route manager

The NX-SDK also supports Python bindings.

For more information on Cisco NX-SDK, go to <https://github.com/CiscoDevNet/NX-SDK> where you can find release and documentation information. Click the `versions.md` link to get information on features and details on each supported release.

Requirements

The NX-SDK has the following requirements:

- Docker
- A Linux environment (either Ubuntu 14.04 or higher, or CentOS 6.7 or higher).
- Cisco SDK (optional) build environment.



Note The Cisco SDK is required to start applications in VSH. VSH requires that all applications be installed through RPMs, which requires that you build them in the Cisco SDK.

The Cisco SDK is not required for Python applications.

The Cisco SDK is not required for C++ application, but is still recommended. Using `g++` to build applications and then copying the built files to the switch may pose stability risks as `g++` is not supported.

Install the NX-SDK

Procedure

Step 1 **Note** The Cisco SDK is required for applications that are started in VSH.

The Cisco SDK is optional for applications that are started in Bash.

(Optional) Build the Cisco SDK RPM to persist on switch reloads and from standby mode.

- a) Pull the Docker image for Ubuntu 14.04+ or Centos 6.7+ from <https://hub.docker.com/r/dockercisco/nxsdk>.
- b) Source for a 32-bit environment:

Example:

```
export ENXOS_SDK_ROOT=/enxos-sdk
cd $ENXOS_SDK_Root
source environment-setup-x86-linux
```

Step 2 Clone the NX-SDK toolkit from <https://github.com/CiscoDevNet/NX-SDK.git>.

Example:

```
git clone https://github.com/CiscoDevNet/NX-SDK.git
```

What to do next

You can find the following references to the API in `$PWD/nxsdk` and includes the following:

- The NX-SDK public C++ classes and APIs,
- Example applications, and
- Example Python applications.

Building and Packaging C++ Applications

The following instructions describe how to build and package your custom C++ NX-OS application.

Procedure

Step 1 Build your application files.

a) Building a C++ application requires adding your source files to the `Makefile`

Example:

The following example uses the `customCliApp.cpp` file from `/examples`.

```
...
##Directory Structure
...
EXNXSDK_BIN:= customCliApp
...
```

b) Build the C++ application using the `make` command.

Example:

```
$PWD/nxsdk# make clean
$PWD/nxsdk# make all
```

Step 2 (Optional) Package your application.

Autogenerate RPM Package

Custom RPM packages for your applications must run on VSH and allow you to specify whether a given application persists on switch reloads or switchovers. Use the following to create a custom specification file for your application.

Note RPM packaging is required to be done within the provided ENXOS Docker image.

a) Use the `rpm_gen.py` script to auto-generate RPM package for a custom application.

Example:

Specify the `-h` option of the script to display the usages of the script.

```
/NX-SDK# python scripts/rpm_gen.py -h
```

b) By default, `NXSDK_ROOT` is set to `/NX-SDK`. If `NX-SDK` is installed in another location other than the default, then you must set `NXSDK_ROOT` env to the appropriate location for the script to run correctly.

Example:

```
export NXSDK_ROOT=<absolute-path-to-NX-SDK>
```

Example of autogenerate RPM package for C++ App `examples/customCliApp.cpp`

```
/NX-SDK/scripts# python rpm_gen.py CustomCliApp
#####
Generating rpm package...

Executing(%prep): /bin/sh -e /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
```

```

+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ exit 0
Executing(%build): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ exit 0
Executing(%install): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ /bin/rm -rf
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root

+ /bin/mkdir -p
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root//isan/bin

+ cp -R /NX-SDK/bin /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/..
/../../var/tmp/customCliApp-root//isan/bin
+ exit 0
Processing files: customCliApp-1.0.x86_64
Requires: libc.so.6 libc.so.6(GLIBC 2.0) 3.0) Libc.so.6(GLIBC_2.1.3) libdl.so.2 libgcc_s.so.1
libgcc_s.so.1(GCC_3.0) libm.so.6 libnxsdk.so libstdc++.so.6 libstdc++.so.6 (CXXABI 1.3)
libstdc++.so.6(GLIBCXX 3.4) libstdc++.so.6(GLIBCXX_3.4.14) rtld(GNU HASH)
Checking for unpackaged file(s):
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/check-files
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root
Wrote: /enxos-sdk/sysroots/X86_64-wrlinuxsdk-linux/usr/src/rpm/SRPMs/customCliApp-1.0.src-rpm

Wrote:
/enxos-sdk/sysroots/X86_64-wrlinuxsdk-linux/usr/src/rpm/RPMS/x86_64/customCliApp-1.0.x86_64.rpm
Executing($clean): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ /bin/rm -rf
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root

RPM package has been built
#####

SPEC file: /NX-SDK/rpm/SPECS/customCliApp.spec
RPM file : /NX-SDK/rpm/RPMS/customCliApp-1.0.x86_64.rpm

```

Manually generate RPM Package

Custom RPM packages for your applications are required to run on VSH and allow you to specify whether a given application persists on switch reloads or system switchovers. Use the following steps to create a custom specification file (*.spec) for your application.

- a) Export the Cisco SDK RPM source to \$RPM_ROOT.

Example:

```
export RPM_ROOT=$ENXOS_SDK_ROOT/sysroots/x86_64-wrlinuxsdk-linux/usr/src/rpm
```

- b) Enter the \$RPM_ROOT directory.

Example:

```
ls $RPM_ROOT (BUILD RPMS SOURCES SPECS SRPMS)
```

- c) Create/edit your application-specific *.spec file.

Refer to the `customCliApp.spec` file in the `/rpm/SPECS` directory for an example specification file.

Note We recommend installing application files to `/isan/bin/nxsdk` on the switch as per the example `customCliApp.spec` file.

Example:

```
vi $RPM_ROOT/SPECS/<application>.spec
```

d) Build your RPM package.

Example:

```
rpm -ba $RPM_ROOT/SPECS/<application>.spec
```

A successful build generates an RPM file in `$RPMS_ROOT/RPMS/x86_64/`

Installing and Running Custom Applications

You can install applications by copying binaries to the switch, or installing unpacking the binaries from the RPM package.



Note Only custom applications that are installed from RPM packages can persist on switch reload or system switchovers. We recommend reserving copying binaries to the switch for simple testing purposes.

Before you begin

The switch must have the NX-SDK enabled before running any custom application. Run **feature nxsdk** on the switch.

Procedure

Step 1 Install your application using either VSH or Bash.

To install your application using VSH, perform the following:

a) Add the RPM package to the installer.

Example:

```
switch(config)# install add bootflash:<app-rpm-package>.rpm
```

b) After installation, check if the RPM is listed as inactive.

Example:

```
switch(config)# show install inactive
```

c) Activate the RPM package.

Example:

```
switch(config)# install activate <app-rpm-package>
```

d) After activation, check if the RPM is listed as active.

Example:

```
switch(config)# show install active
```

To install your application using Bash, run the following commands:

```
switch(config)# run bash sudo su
bash# yum install /bootflash/<app-rpm-package>.rpm
```

Step 2

Start your application.

C++ applications can run from VSH or Bash.

- To run a C++ application in VSH, run the **nxsdk** command:

```
switch(config)# nxsdk service-name /<install directory>/<application>
```

Note If the application is installed in `/isan/bin/nxsdk`, the full file path is not required. You can use the **nxsdk service-name app-name** form of the command.

- To run a C++ application in Bash, start Bash then start the application.

```
switch(config)# run bash sudo su
bash# <app-full-path> &
```

Python applications can run from VSH or Bash.

- To run a Python application from VSH, run the **nxsdk** command:

```
switch(config)# nxsdk service-name <app-full-path>
```

Note The Python application must be made executable to start from VSH:

- Run the **chmod +x <app-full-path>** command.
- Add `#!/isan/bin/nxpython` to the first link of your Python application.

- To run a Python application from Bash:

```
switch(config)# run bash sudo su
bash# /isan/bin/python <app-full-path>
```

Note Use `/isan/bin/python` to run Python applications in Bash.

Step 3

Run **show nxsdk internal service** to verify that your application is running.

Example:

```
switch(config)# show nxsdk internal service
```

```
NXSDK Started/Temp unavailabe/Max services : 2/0/32
NXSDK Default App Path      : /isan/bin/nxsdk
NXSDK Supported Versions   : 1.0 1.5 1.7.5
```

Service-name	Base App	Started(PID)	Version	RPM Package
/isan/bin/capp1	nxsdk_app2	VSH(25270)	1.7.5	capp1-1.0.x86_64
/isan/bin/TestApp.py	nxsdk_app3	BASH(27823)	-	-

Step 4

Stop your application.

You can stop your application in the following ways:

- To stop all NX-SDK applications, run the **no feature nxsdk** command.
- To stop a specific application in VSH, run:
switch(config)# no nxsdk service-name /<install directory>/<application>
- To stop a specific application in Bash, run:
bash# <application> stop-event-loop

Step 5 Uninstall your application.

To uninstall the RPM from the switch using VSH, perform the following:

- a) Deactivate the active RPM package.

Example:

```
switch# install deactivate <app-rpm-package>
```

- b) Verify that the package is deactivated.

Example:

```
switch# show install inactive
```

- c) Remove the RPM package.

Example:

```
switch# install remove <app-rpm-package>
```

To uninstall the RPM from the switch using Bash, run the `yum remove <app-full-path>` command.



PART **III**

NX-API

- [NX-API CLI, on page 109](#)
- [NX-API REST, on page 129](#)
- [NX-API Developer Sandbox, on page 131](#)



CHAPTER 11

NX-API CLI

This chapter contains the following topics:

- [About NX-API CLI, on page 109](#)
- [Using NX-API CLI, on page 110](#)
- [JSON and XML Structured Output, on page 124](#)

About NX-API CLI

NX-API CLI is an enhancement to the Cisco Nexus 3400-S Series CLI system, which supports XML output. NX-API CLI also supports JSON output format for specific commands.

On Cisco Nexus devices, command-line interfaces (CLIs) are run only on the device. NX-API CLI improves the accessibility of these CLIs by making them available outside of the switch by using HTTP or HTTPS. You can use this extension to the existing Cisco Nexus CLI system on the Cisco Nexus 3400-S Series devices. NX-API CLI supports **show** commands, configurations, and Linux Bash.

NX-API CLI supports JSON-RPC.

Transport

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

The NX-API feature is enabled by default on HTTPS port 443. HTTP port 80 is disabled.

NX-API is also supported through UNIX Domain Sockets for applications running natively on the host or within Guest Shell.

The NX-API backend uses the Nginx HTTP server. The Nginx process, and all its children processes, are under Linux cgroup protection which caps the CPU and memory usage. 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 3400-S 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 does not support insecure HTTP by default.
- NX-API does not support weak TLSv1 protocol by default.

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 CLI

The commands, command type, and output type for the Cisco Nexus 3400-S switches are entered using NX-API. NX-API encodes the CLIs into the body of an HTTP or HTTPS POST request. 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 123](#).

NX-API CLI is enabled by default for local access. The remote HTTP access is disabled by default.

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

- Enable the management interface.

```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 192.0.20.123/24
switch(config)# vrf context management
switch(config)# ip route 10.0.113.1/0 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"
      }
    }
  }
}

```



Note There is a known issue where an attempt to delete a user might fail. The result is an error similar to the following appearing approximately every 12 hours:

```
user delete failed for username:userdel: user username is currently logged in - securityd
```

This issue might occur in a scenario where you try to delete a user who is still logged into a switch through NX-API. Enter the following command in this case to try to log the user out first:

```
switch(config)# clear user username
```

Then try to delete the user again. If the issue persists after attempting this workaround, contact Cisco TAC for further assistance.

Escalate Privileges to Root on NX-API

For NX-API, the privileges of an Admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- Only an Admin user can escalate privileges to root.
- Escalation to root is password-protected.

The following examples show how an Admin escalates privileges to root and how to verify the escalation. After becoming root, the **whoami** command shows you as Admin; however, the Admin account has all the root privileges.

First example:


```

<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>

```

Second example:

```

<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>

```

Sample NX-API Scripts

The sample scripts demonstrate how a script is used with NX-API.

- Cable Checker (check_cable.py)
- Cable Checker Blueprint (connectivity.json)
- Using NX-API over UDS (rest_client.py)

NX-API Management Commands

The following table shows the CLI commands that can manage and enable through the NX-API.

Table 4: NX-API Management Commands

NX-API Management Command	Description
feature nxapi	Enables NX-API.
no feature nxapi	Disables NX-API.
nxapi {http https} port <i>port</i>	Specifies a port.
no nxapi {http https}	Disables HTTP or HTTPS.
show nxapi	Displays port information.
nxapi certificate {httpsrct certfile httpskey keyfile} <i>filename</i>	<p>Specifies the upload of the following:</p> <ul style="list-style-type: none"> • HTTPS certificate when <code>httpsrct</code> is specified. • HTTPS key when <code>httpskey</code> is specified. <p>Example of an HTTPS certificate:</p> <pre>nxapi certificate httpsrct certfile bootflash:cert.crt</pre> <p>Example of an HTTPS key:</p> <pre>nxapi certificate httpskey keyfile bootflash:privkey.key</pre>
nxapi certificate enable	Enables a certificate.
nxapi ssl-ciphers weak	Weak ciphers are disabled by default. Running this command changes the default behavior and enables the weak ciphers for NGINX. The no form of the command changes it to the default (by default, the weak ciphers are disabled).
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	TLS1.0 is disabled by default. Running this command enables the TLS versions that are specified in the string, including the TLS1.0 that was disabled by default, if necessary. The no form of the command changes it to the default (by default, only TLS1.1 and TLS1.2 are enabled).
nxapi use-vrf <i>vrf</i>	Specifies the default VRF, management VRF, or named VRF.
ip netns exec management iptables	<p>Implements any access restrictions and can be run in management VRF.</p> <p>Note You must enable feature bash-shell and then run the command from Bash Shell. For more information on Bash Shell, see the chapter on Bash.</p> <p><i>Iptables is a command-line firewall utility that uses policy chains to allow or block traffic and almost always comes pre-installed on any Linux distribution.</i></p>

Following is an example of a successful upload of an HTTPS certificate:

```
switch(config)# nxapi certificate https crt certfile certificate.crt
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

Following is an example of a successful upload of an HTTPS key:

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

In some situations, you can get an error message saying that the certificate is invalid:

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
Ngix certificate invalid.
switch(config)#
```

This error can occur if the key file is encrypted. In that case, you must decrypt the key file before you can install it. You might have to go into Guest Shell to decrypt the key file, as shown in the following example:

```
switch(config)# guestshell
[b3456@guestshell ~]$
[b3456@guestshell bootflash]$ /bin/openssl rsa -in certfilename.net.pem -out clearkey.pem

Enter pass phrase for certfilename.net.pem:
writing RSA key
[b3456@guestshell bootflash]$
[b3456@guestshell bootflash]$ exit
switch(config)#
```

See the Guest Shell chapter in this document for more information on Guest Shell.

If this was the reason for the issue, you should now be able to successfully install the certificate:

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

Working with Interactive Commands Using NX-API

To disable confirmation prompts on interactive commands and avoid time outs with an error code 500, prepend interactive commands with **terminal dont-ask**. Use ; to separate multiple interactive commands, where each ; is surrounded with single blank characters.

Following are several examples of interactive commands that use **terminal dont-ask** to avoid timing out with an error code 500:

```
terminal dont-ask ; reload module 21
terminal dont-ask ; system mode maintenance
```

NX-API Request Elements

NX-API sends request elements 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 5: NX-API Request Elements for XML or JSON Format

NX-API Request Element	Description
version	Specifies the NX-API version.
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> • cli_show CLI show commands that expect structured output. If the command does not support XML output, an error message is returned. • cli_show_array CLI show commands that expect structured output. Only for show commands. Similar to cli_show, but with cli_show_array, data is returned as a list of one element, or an array, within square brackets []. • 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. <p>Note</p> <ul style="list-style-type: none"> • Each command is only executable with the current user's authority. • The pipe operation is supported in the output when the message type is ASCII. If the output is in XML format, the pipe operation is not supported. • A maximum of 10 consecutive show commands are supported. If the number of show commands exceeds 10, the 11th and subsequent commands are ignored. • No interactive commands are supported.

NX-API Request Element	Description				
<i>chunk</i>	<p>Some show commands can return a large amount of output. For the NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for show commands.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="824 485 1523 596"> <tr> <td data-bbox="824 485 938 539">0</td> <td data-bbox="938 485 1523 539">Do not chunk output.</td> </tr> <tr> <td data-bbox="824 539 938 596">1</td> <td data-bbox="938 539 1523 596">Chunk output.</td> </tr> </table> <p>Note Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.</p> <p>The output message format is XML. (XML is the default.) Special characters, such as < or >, are converted to form a valid XML message (< is converted into &lt; > is converted into &gt;).</p> <p>You can use XML SAX to parse the chunked output.</p> <p>Note When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</p>	0	Do not chunk output.	1	Chunk output.
0	Do not chunk output.				
1	Chunk output.				
<i>rollback</i>	<p>Valid only for configuration CLIs, not for show commands. Specifies the configuration rollback options. Specify one of the following options.</p> <ul style="list-style-type: none"> • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in. <p>Note The rollback element is available in the cli_conf mode when the input request format is XML or JSON.</p>				
<i>sid</i>	<p>The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a <i>sid</i> to match the <i>sid</i> of the previous response message.</p>				

NX-API Request Element	Description						
<i>input</i>	<p>Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, show commands are cli_show message type and are not supported in cli_conf mode.</p> <p>Note Except for bash, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.)</p> <p>Prepend commands with <code>terminal dont-ask</code> to avoid timing out with an error code 500. For example:</p> <pre>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</pre> <p>For bash, multiple commands are separated with ";". (The ; is not surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="786 863 1490 1087"> <tbody> <tr> <td data-bbox="786 863 911 936">cli_show</td> <td data-bbox="911 863 1490 936">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="786 936 911 1010">cli_conf</td> <td data-bbox="911 936 1490 1010">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="786 1010 911 1087">bash</td> <td data-bbox="911 1010 1490 1087">cd /bootflash;mkdir new_dir</td> </tr> </tbody> </table>	cli_show	show version ; show interface brief ; show vlan	cli_conf	interface Eth4/1 ; no shut ; switchport	bash	cd /bootflash;mkdir new_dir
cli_show	show version ; show interface brief ; show vlan						
cli_conf	interface Eth4/1 ; no shut ; switchport						
bash	cd /bootflash;mkdir new_dir						
<i>output_format</i>	<p>The available output message formats are the following:</p> <table border="1" data-bbox="786 1163 1490 1276"> <tbody> <tr> <td data-bbox="786 1163 1024 1220">xml</td> <td data-bbox="1024 1163 1490 1220">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="786 1220 1024 1276">json</td> <td data-bbox="1024 1220 1490 1276">Specifies output in JSON format.</td> </tr> </tbody> </table> <p>Note The Cisco Nexus 3400-S Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.</p> <p>To manage the computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported.</p> <p>The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).</p>	xml	Specifies output in XML format.	json	Specifies output in JSON format.		
xml	Specifies output in XML format.						
json	Specifies output in JSON format.						

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

Table 6: NX-API Request Elements for JSON-RPC Format

NX-API Request Element	Description
<i>jsonrpc</i>	A string specifying the version of the JSON-RPC protocol. Version must be 2.0.
<i>method</i>	A string containing the name of the method to be invoked. NX-API supports either: <ul style="list-style-type: none"> • cli—show or configuration commands • cli_ascii—show or configuration commands; output without formatting • cli_array—only for show commands; similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [].
<i>params</i>	A structured value that holds the parameter values used during the invocation of a method. It must contain the following: <ul style="list-style-type: none"> • cmd—CLI command • version—NX-API request version identifier
<i>rollback</i>	Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options. <ul style="list-style-type: none"> • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores the failed CLI and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
<i>id</i>	An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the <i>id</i> parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, <i>id</i> : 1

NX-API Response Elements

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

Table 7: 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 <code>cli_show</code> or <code>cli_show_ascii</code> , each command output is enclosed by a single output tag. When the message type is <code>cli_conf</code> or <code>bash</code> , there is a single output tag for all the commands because <code>cli_conf</code> and <code>bash</code> commands require context.
output	Tag that encloses the output of a single command output. For <code>cli_conf</code> and <code>bash</code> 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.
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 that is associated with the returned error code.

Restricting Access to NX-API

There are two methods for restricting HTTP and HTTPS access to a device: ACLs and iptables. The method that you use depends on whether you have configured a VRF for NX-API communication using the `nxapi use-vrf <vrf-name>` CLI command.

Use ACLs to restrict HTTP or HTTPS access to a device only if you have not configured a VRF for NX-API communication. For information about configuring ACLs, see the *Cisco Nexus 3400-S NX-OS Security Configuration Guide*.

If you have configured a VRF for NX-API communication, however, ACLs will not restrict HTTP or HTTPS access. Instead, create a rule for an iptable. For more information about creating a rule, see [Updating an iptable](#), on page 121.

Updating an iptable

An iptable enables you to restrict HTTP or HTTPS access to a device when a VRF is configured for NX-API communication. This section demonstrates how to add, verify, and remove rules for blocking HTTP and HTTPS access to an existing iptable.

Procedure

Step 1 To create a rule that blocks HTTP access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 80 -j DROP
```

Step 2 To create a rule that blocks HTTPS access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 443 -j DROP
```

Step 3 To verify the applied rules:

```
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:http
DROP      tcp  --  anywhere              anywhere              tcp dpt:https
DROP      tcp  --  anywhere              anywhere             

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Step 4 To create and verify a rule that blocks all traffic with a 10.155.0.0/24 subnet to port 80:

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j
DROP
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:http
DROP      tcp  --  10.155.0.0/24        anywhere              tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Step 5 To remove and verify previously applied rules:

This example removes the first rule from INPUT.

```
bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
```

```
target      prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

What to do next

The rules in iptables are not persistent across reloads when you modify them in the Bash shell. To make the rules persistent, see [Making an Iptable Persistent Across Reloads, on page 122](#).

Making an Iptable Persistent Across Reloads

The rules in iptables are not persistent across reloads when you modify them in the Bash shell. This section explains how to make a modified iptable persistent across a reload.

Before you begin

You have modified an iptable.

Procedure

Step 1 Create a file called `iptables_init.log` in the `/etc` directory with full permissions:

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

Step 2 Create a startup script called `iptables_init` in the `/etc/init.d` directory with the following set of commands:

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          iptables_init
# Required-Start:
# Required-Stop:
# Default-Start:    2 3 4 5
# Default-Stop:
# Short-Description: init for iptables
# Description:      sets config for iptables
#                   during boot time

### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
```

```

}
case "$1" in
  start)
    start_script
    ;;
  stop)
    ;;
  restart)
    sleep 1
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac
exit 0

```

Step 3 Set the appropriate permissions to the startup script:

```
bash-4.3# chmod 777 /etc/init.d/iptables_init
```

Step 4 Set the iptables_init startup script to on with the chkconfig utility:

```
bash-4.3# chkconfig iptables_init on
```

The iptables_init startup script now executes each time that you perform a reload, making the iptable rules persistent.

Table of NX-API Response Codes

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 8: 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.

JSON and XML Structured Output

The NX-OS supports redirecting the standard output of various **show** commands in the following structured output formats:

- XML
- JSON
- JSON Pretty, which makes the standard block of JSON-formatted output easier to read.

Converting the standard NX-OS output to JSON, JSON Pretty, or XML format occurs on the NX-OS CLI by "piping" the output to a JSON or XML interpreter. For example, you can issue the **show ip access** command with the logical pipe (|) and specify JSON, JSON Pretty, or XML, and the NX-OS command output will be properly structured and encoded in that format. This feature enables programmatic parsing of the data and supports streaming data from the switch through software streaming telemetry. Most commands in Cisco NX-OS support JSON, JSON Pretty, and XML output.

Selected examples of this feature follow.

About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard that is designed for human-readable data and is an alternative to XML. JSON originally evolved from JavaScript, but it is a language-independent data format. Command output from NX-OS supports JSON and JSON Pretty format.

The Cisco Nexus 3400-S supports JSON CLI Execution.

Nearly all modern programming languages support two primary Data Structures in some way. These Data Structures are:

- Ordered List :: Array
- Unordered List (Name/Value pair) :: Objects

Also, through the NX-OS Sandbox, you can access JSON or XML output for a **show** command.

CLI Execution

```
switch-1# show cdp neighbors | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "83886080", "device_id": "SW-SPARSHA-SAVBU-F10", "intf_id": "mgmt0", "ttl": "148", "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco WS-C2960 S-48TS-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device_id": "BLR-VXLAN-NPT-CR-178 (FOC1745R01W)", "intf_id": "Ethernet1/1", "ttl": "166", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Dispute"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
switch-1#
```

Examples of XML and JSON Output

This section documents selected examples of NX-OS commands that are displayed as XML and JSON output.

This example shows how to display the unicast and multicast routing entries in hardware tables in JSON format:

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096", "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "used_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_total": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "used_mcast_oif1": "2", "used_host_in_host_total": "13", "used_host4_in_host": "12", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table": "0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"}
switch(config)#
```

This example shows how to display the unicast and multicast routing entries in hardware tables in XML format:

```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:fib">
  <nf:data>
    <show>
      <hardware>
        <profile>
          <status>
            <__XML_OPT_Cmd_dynamic_tcam_status>
              <__XML_OPT_Cmd_dynamic_tcam_status__readonly__>
                <__readonly__>
                  <total_lpm>8191</total_lpm>
                  <total_host>8192</total_host>
                  <total_lpm>1024</total_lpm>
                  <max_host4_limit>4096</max_host4_limit>
                  <max_host6_limit>2048</max_host6_limit>
                  <max_mcast_limit>2048</max_mcast_limit>
                  <used_lpm_total>9</used_lpm_total>
                  <used_v4_lpm>6</used_v4_lpm>
                  <used_v6_lpm>3</used_v6_lpm>
                  <used_v6_lpm_128>1</used_v6_lpm_128>
                  <used_host_lpm_total>0</used_host_lpm_total>
                  <used_host_v4_lpm>0</used_host_v4_lpm>
                  <used_host_v6_lpm>0</used_host_v6_lpm>
                  <used_mcast>0</used_mcast>
                  <used_mcast_oif1>2</used_mcast_oif1>
                  <used_host_in_host_total>13</used_host_in_host_total>
                  <used_host4_in_host>12</used_host4_in_host>
                  <used_host6_in_host>1</used_host6_in_host>
                  <max_ecmp_table_limit>64</max_ecmp_table_limit>
                  <used_ecmp_table>0</used_ecmp_table>
                  <mfib_fd_status>Disabled</mfib_fd_status>
                  <mfib_fd_maxroute>0</mfib_fd_maxroute>
                  <mfib_fd_count>0</mfib_fd_count>
                </__readonly__>
              </__XML_OPT_Cmd_dynamic_tcam_status__readonly__>
            </__XML_OPT_Cmd_dynamic_tcam_status>
          </status>
        </profile>
      </hardware>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

This example shows how to display LLDP timers that are configured on the switch in JSON format:

```
switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_mplier": "4", "notification_interval": "5"}
switch(config)#
```

This example shows how to display LLDP timers that are configured on the switch in XML format:

```
switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:fib">
  <nf:data>
    <show>
      <lldp>
        <timers>
          <ttl>120</ttl>
          <reinit>2</reinit>
          <tx_interval>30</tx_interval>
          <tx_delay>2</tx_delay>
          <hold_mplier>4</hold_mplier>
          <notification_interval>5</notification_interval>
        </timers>
      </lldp>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

```
ww.cisco.com/nxos:1.0:lldp">
<nf:data>
  <show>
    <lldp>
      <timers>
        <__XML__OPT_Cmd_lldp_show_timers__readonly__>
          <__readonly__>
            <ttl>120</ttl>
            <reinit>2</reinit>
            <tx_interval>30</tx_interval>
            <tx_delay>2</tx_delay>
            <hold_mplier>4</hold_mplier>
            <notification_interval>5</notification_interval>
          </__readonly__>
        </__XML__OPT_Cmd_lldp_show_timers__readonly__>
      </timers>
    </lldp>
  </show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```




CHAPTER 12

NX-API REST

This chapter contains the following topics:

- [About NX-API REST, on page 129](#)

About NX-API REST

NX-API REST

On Cisco Nexus 3400-S devices, you perform configuration tasks by using command-line interfaces (CLIs) that run only on the device. NX-API REST improves the accessibility of the Nexus configuration by providing HTTP or HTTPS APIs that:

- Make specific CLIs available outside of the switch.
- Enable configurations that would require issuing many CLI commands by combining configuration actions in relatively few HTTP or HTTPS operations.

NX-API REST supports **show** commands, basic and advanced switch configurations, and Linux Bash.

NX-API REST uses HTTP or HTTPS as its transport. CLIs are encoded into the HTTP or HTTPS POST body. The NX-API REST backend uses the Nginx HTTP server. The Nginx process, and all its children processes, are under Linux cgroup protection where the CPU and memory usage is capped. If the Nginx resource usage exceeds the cgroup limitations, the Nginx process is restarted and restored.

For more information about the Cisco Nexus 3400-S Series NX-API REST SDK, see <https://developer.cisco.com/docs/nx-os-n3k-n9k-api-ref/>.



CHAPTER 13

NX-API Developer Sandbox

This chapter contains the following topics:

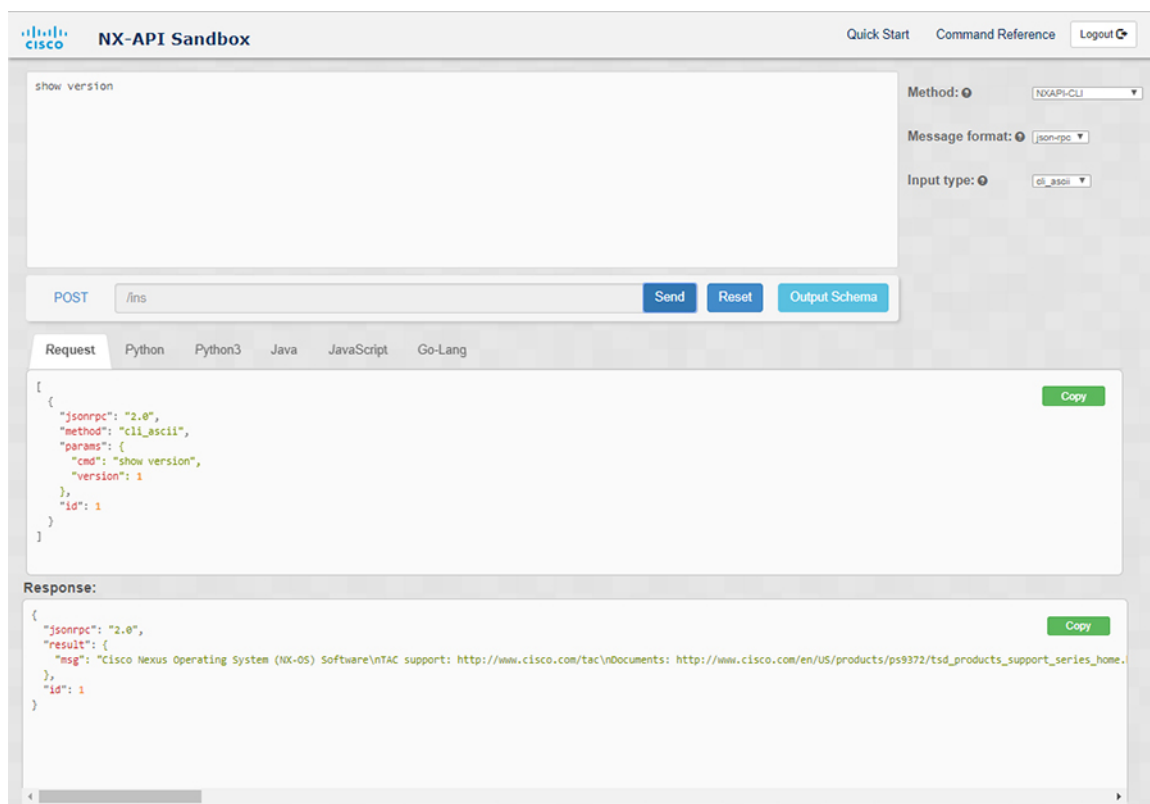
- [About the NX-API Developer Sandbox, on page 131](#)
- [Guidelines and Limitations, on page 132](#)
- [Enabling and Accessing the Developer Sandbox, on page 133](#)
- [Configuring the Message Format and Input Type, on page 133](#)
- [Using the Developer Sandbox, on page 135](#)

About the NX-API Developer Sandbox

The NX-API Developer Sandbox is a web form hosted on the switch. It translates NX-OS CLI commands into equivalent XML or JSON payloads, and converts NX-API REST payloads into their CLI equivalents.

The web form is a single screen with three panes — Command (top pane), Request, and Response — as shown in the figure.

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



Controls in the Command pane allow you to choose a message format for a supported API, such as NX-API REST, and a command type, such as XML or JSON. The available command type options vary depending on the selected message format.

When you type or paste one or more CLI commands into the Command pane, the web form converts the commands into an API payload, checking for configuration errors, and displays the resulting payload in the Request pane. If you then choose to post the payload directly from the Sandbox to the switch, using the POST button in the Command pane, the Response pane displays the API response.

Conversely, when you type an NX-API REST designated name (DN) and payload into the Command pane and select the **NX-API-REST (DME)** Method format and the **model** Input type, Developer Sandbox checks the payload for configuration errors, then the Response pane displays the equivalent CLIs.

Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled.
- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.

- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.

Enabling and Accessing the Developer Sandbox

By default, NX-API is disabled on the switch, as is the Sandbox. This procedure shows you how to enable both and how to access the Sandbox with a browser.

Before you begin

Configure the management port of the switch.

Procedure

-
- Step 1** Enable the NX-API **nxapi** feature and the sandbox using the following commands.
- ```
switch# configure terminal
switch(config)# feature nxapi
```
- Step 2** (Optional) To change the HTTP or HTTPS port number, enter the **nxapi http[s] port port-number** command.
- Example:**
- ```
switch# configure terminal
switch(config)# nxapi http port 80
switch(config)# nxapi https port 443
```
- Step 3** Open a browser and enter **http[s]://management-ip-address** (or **http[s]://management-ip-address:port-number**, if you configured with a specific port number in the previous step) to launch the NX-API Developer Sandbox.
- Example:**
- If the management IP address of the switch is 192.0.20.123, browse to `https://192.0.20.123`.
- Note** Cisco recommends that you use the Chrome browser, release 69.0.3497.100 (64-bit), or later, to access the NX-API Developer Sandbox.
-

Configuring the Message Format and Input Type

The **Method**, **Message format**, and **Input type** are configured in the upper right corner of the Command pane (the top pane). For **Method**, choose the format of the API protocol that you want to use. The Developer Sandbox supports the following API protocols:

Table 9: NX-OS API Protocols

Protocol	Description
NXAPI-CLI	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML or a JSON payload.

Protocol	Description
NXAPI-REST (DME)	Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. For more information about the Cisco Nexus 3400-S Series NX-API REST SDK, see https://developer.cisco.com/site/cisco-nexus-nx-api-references/ .
RESTCONF (Yang)	The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.

When the **Method** has been chosen, a set of **Message format** and **Input type** options are presented. The **Message format** setting can constrain the input CLI and can determine the **Request** and **Response** format. The options vary depending on the **Method** selection.

For each **Message format**, the following table describes the **Input type** options:

Table 10: Command Types

Method	Message format	Input type
NXAPI-CLI	json-rpc	<ul style="list-style-type: none"> cli — show or configuration commands cli-ascii — show or configuration commands, output without formatting cli-array — show or configuration commands. Similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [].
NXAPI-CLI	xml	<ul style="list-style-type: none"> cli_show — show commands. If the command does not support XML output, an error message will be returned. cli_show_array — show commands. Similar to cli_show, but with cli_show_array, data is returned as a list of one element, or an array, within square brackets []. cli_show_ascii — show commands, output without formatting cli_conf — configuration commands. Interactive configuration commands are not supported. bash — bash commands. Most non-interactive bash commands are supported. <p>Note The bash shell must be enabled in the switch.</p>

Method	Message format	Input type
NXAPI-CLI	json	<ul style="list-style-type: none"> cli_show — show commands. If the command does not support XML output, an error message will be returned. cli_show_ascii — show commands, output without formatting cli_conf — configuration commands. Interactive configuration commands are not supported. bash — bash commands. Most non-interactive bash commands are supported. <p>Note The bash shell must be enabled in the switch.</p>
NXAPI-REST (DME)		<ul style="list-style-type: none"> cli — configuration commands model — DN and corresponding payload.
RESTCONF (Yang)	<ul style="list-style-type: none"> json — JSON structure is used for payload xml — XML structure is used for payload 	

Output Chunking

In order to handle large show command output, some NX-API message formats support output chunking for show commands. In this case, an **Enable chunk mode** checkbox appears below the **Command Type** control along with a session ID (**SID**) type-in box.

When chunking is enabled, the response is sent in multiple "chunks," with the first chunk sent in the immediate command response. In order to retrieve the next chunk of the response message, you must send an NX-API request with **SID** set to the session ID of the previous response message.

Using the Developer Sandbox

You can use the Developer Sandbox to make multiple conversions, including the following:

- [Using the Developer Sandbox to Convert CLI Commands to Payloads, on page 136](#)
- [Using the Developer Sandbox to Convert from REST Payloads to CLI Commands, on page 139](#)
- [Using the Developer Sandbox to Convert from RESTCONF to json or XML, on page 143](#)

Using the Developer Sandbox to Convert CLI Commands to Payloads



Tip Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window. Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI". Only configuration commands are supported.

Procedure

- Step 1** Configure the **Method**, **Message Format**, and **Input Type** for the API protocol you want to use. For detailed instructions, see [Configuring the Message Format and Input Type, on page 133](#).
- Step 2** Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.
- You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.

- Step 3** Click the **Convert** at the bottom of the top pane.
- If the CLI commands contain no configuration errors, the payload appears in the **Request** pane. If errors are present, a descriptive error message appears in the **Response** pane.


```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

Method: NXAPI-REST (DME)

Input type: model

/api/mo/sys.json

Send Reset Convert

Request Python Python3 Java JavaScript Go-Lang

hostname REST2CLI

Copy

Response:

Copy

Step 4 When a valid payload is present in the **Request** pane, you can click **Send** to send the payload as an API call to the switch.

The response from the switch appears in the **Response** pane.

Warning Clicking **Send** commits the command to the switch, which can result in a configuration or state change.

Logging level `netstack 6`

Method:

Input type:

POST

Request Python Python3 Java JavaScript Go-Lang

```
{
  "topSystem": {
    "children": [
      {
        "ipV4Entity": {
          "children": [
            {
              "ipV4Inst": {
                "attributes": {
                  "loggingLevel": "informational"
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

Copy

Response:

```
{
  "imdata": []
}
```

Copy

Step 5 You can copy the contents of the **Request** or **Response** pane to the clipboard by clicking **Copy** in the pane.

Step 6 You can also convert the request into the following formats by clicking on the appropriate tab in the **Request** pane:

- Python
- Python3
- Java
- JavaScript
- Go-Lang

Using the Developer Sandbox to Convert from REST Payloads to CLI Commands



Tip Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window. Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI". Click on the **DME Documentation** link in the upper right corner of the Sandbox window to go to the NX-API DME Model Reference page.

Click on the **Model Browser** link in the upper right corner of the Sandbox window to access Visore, the Model Browser. Note that you might have to manually enter the IP address for your switch to access the Visore page:

`https://management-ip-address/visore.html.`

Procedure

Step 1 Select **NXAPI-REST (DME)** as the **Method** and **model** as the **Input Type**.

Example:

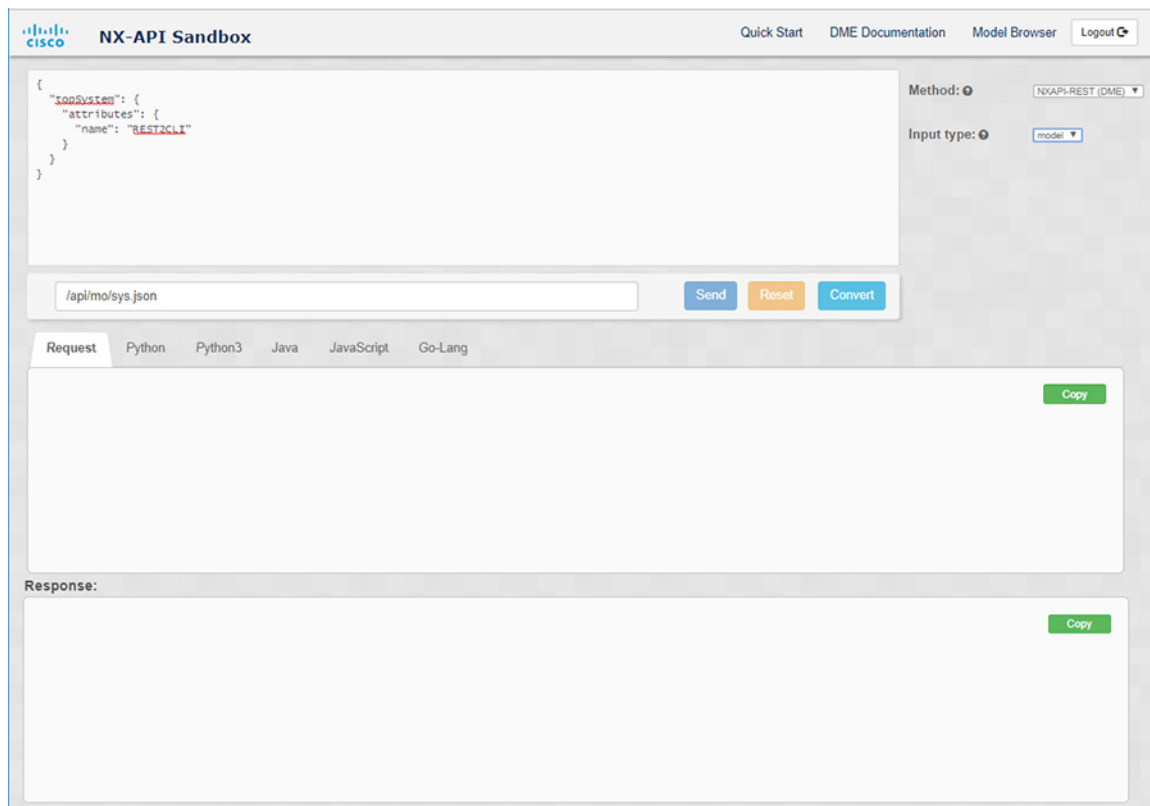
The screenshot shows the NX-API Sandbox interface. At the top, there are navigation links: Quick Start, DME Documentation, Model Browser, and Logout. The main area is divided into two panes. The top pane is for entering the DME payload, with a text box containing "/api/mo/sys.json". To the right of the text box are dropdown menus for Method (set to NXAPI-REST (DME)) and Input type (set to model). Below the text box are buttons for Send, Reset, and Convert. The bottom pane is for the response, with a tab labeled "Request" and other tabs for Python, Python3, Java, JavaScript, and Go-Lang. A "Copy" button is visible in the bottom right corner of the response pane.

Step 2 Enter a DN and payload into the text entry box in the top pane. Then click on the **Convert** button below the top pane.

Example:

For this example, the DN is `/api/mo/sys.json` and the NX-API REST payload is:

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



When you click on the **Convert** button, the CLI equivalent appears in the **CLI** pane as shown in the following image.

The screenshot displays the Cisco NX-API Sandbox interface. At the top left is the Cisco logo and the text "NX-API Sandbox". On the top right, there are navigation links: "Quick Start", "DME Documentation", "Model Browser", and "Logout".

The main area is divided into two sections. The upper section contains a JSON payload in a text area:


```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

 To the right of this area are two dropdown menus: "Method" set to "NX-API-REST (DME)" and "Input type" set to "model". Below the text area is a URL input field containing "/api/mo/sys.json" and three buttons: "Send", "Reset", and "Convert".

The lower section is titled "Request" and has tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing the output "hostname REST2CLI" with a green "Copy" button to its right.

Below the request section is a "Response:" section, which is currently empty, with a green "Copy" button to its right.

Note The Developer Sandbox cannot convert all payloads into equivalent CLIs, even if the Sandbox converted the CLIs to NX-API REST payloads. The following is a list of possible sources of error that can prevent a payload from completely converting to CLI commands:

Table 11: Sources of REST2CLI Errors

Payload Issue	Result
<p>The payload contains an attribute that does not exist in the MO.</p> <p>Example:</p> <pre>api/mo/sys.json { "topSystem": { "children": [{ "interfaceEntity": { "children": [{ "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": "totallyFake" } } }] } }] } }</pre>	<p>The Error pane will return an error related to the attribute.</p> <p>Example:</p> <p>CLI</p> <p>Error unknown attribute 'fakeattribute' in element 'l1PhysIf'</p>
<p>The payload includes MOs that aren't yet supported for conversion:</p> <p>Example:</p> <pre>api/mo/sys.json { "topSystem": { "children": [{ "dhcpEntity": { "children": [{ "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" } } }] } }] } }</pre>	<p>The Error Pane will return an error related to the unsupported MO.</p> <p>Example:</p> <p>CLI</p> <p>Error The entire subtree of "sys/dhcp" is not converted.</p>

Using the Developer Sandbox to Convert from RESTCONF to json or XML



- Tip** Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window.
- Click on the **Yang Documentation** link in the upper right corner of the Sandbox window to go to the Model Driven Programmability with Yang page.
- Click on the **Yang Models** link in the upper right corner of the Sandbox window to access the YangModels GitHub site.

Procedure

- Step 1** Select **RESTCONF (Yang)** as the **Method** and either **json** or **xml** as the **Message format**.

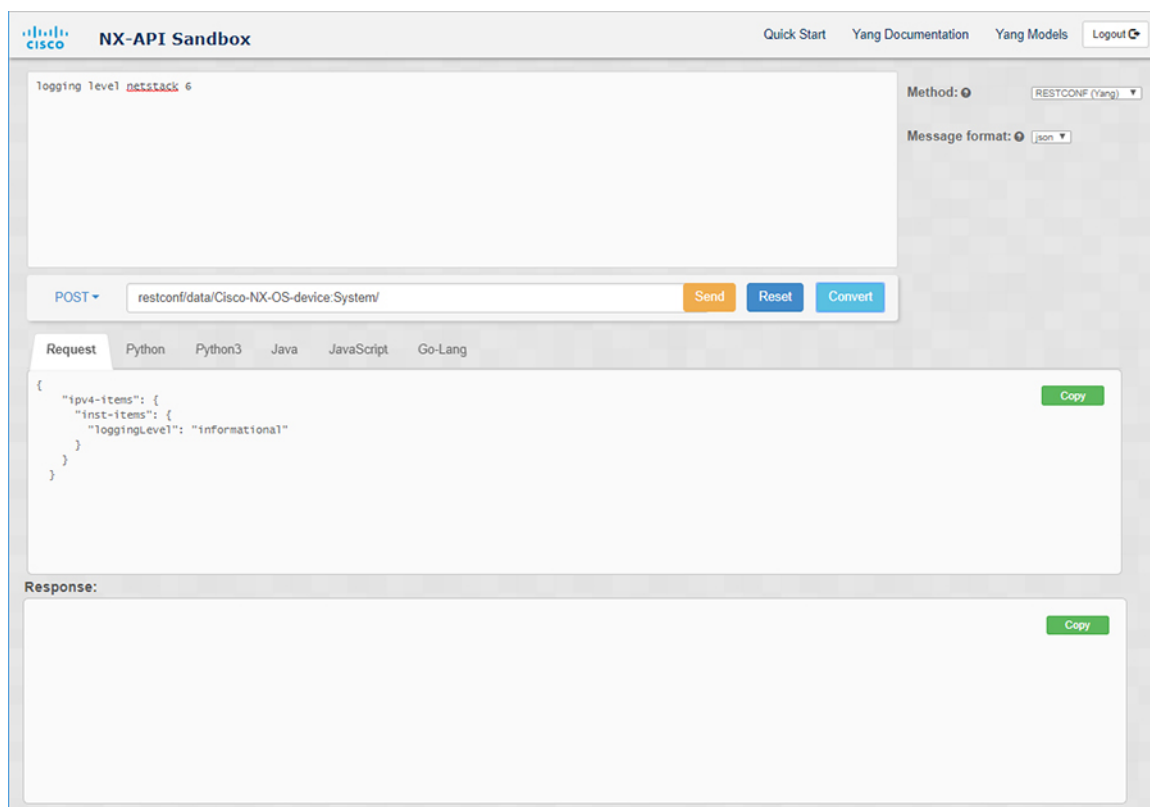
Example:

The screenshot displays the NX-API Sandbox interface. At the top, there are navigation links: "Quick Start", "Yang Documentation", "Yang Models", and "Logout". The main area is divided into two panes. The top pane contains a text entry box with the command "Logging level netstack 6". To the right of this box are two dropdown menus: "Method" set to "RESTCONF (Yang)" and "Message format" set to "json". Below the text entry box is a "POST" dropdown menu, a text input field containing "restconf/data/Cisco-NX-OS-device:System/", and three buttons: "Send", "Reset", and "Convert". The bottom pane is titled "Request" and contains a large empty text area with a "Copy" button. Below this is a "Response:" section, also with a large empty text area and a "Copy" button.

- Step 2** Enter a command into the text entry box in the top pane, choose a message format, then click on the **Convert** button below the top pane.

Example:

For this example, the command is **logging level netstack 6** and the message format is json:



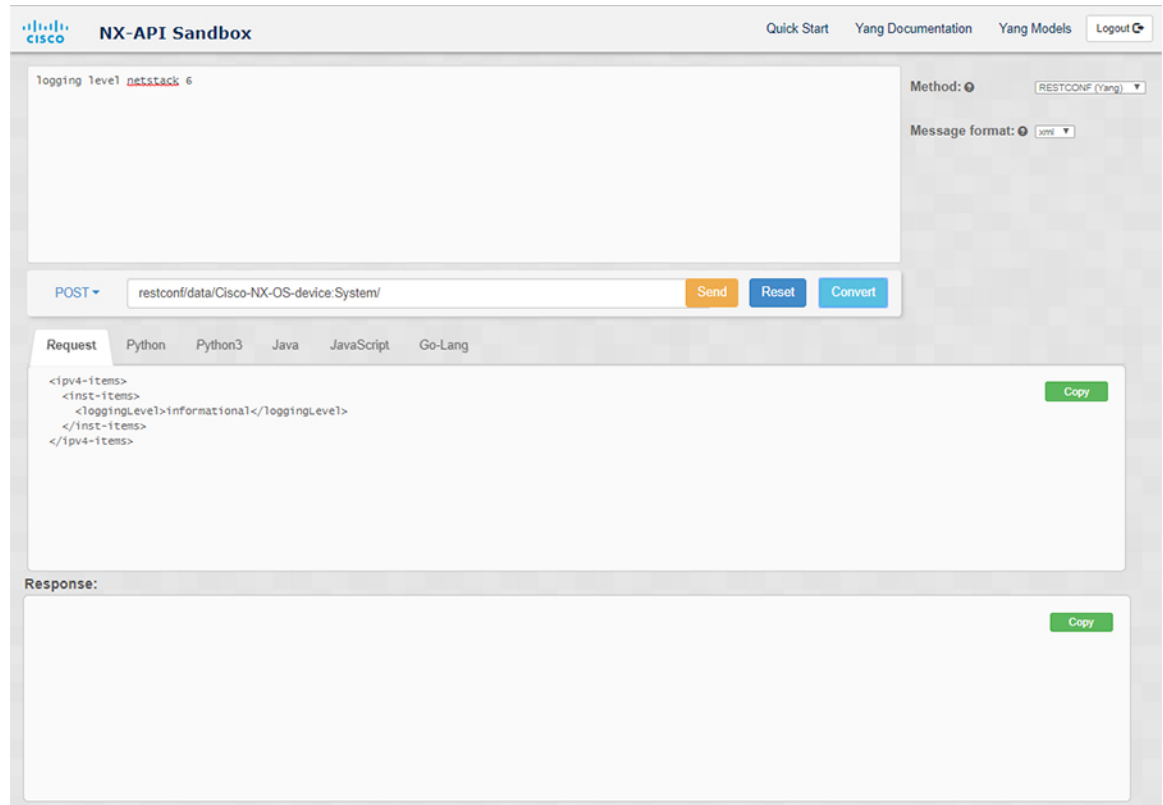
The screenshot displays the Cisco NX-API Sandbox interface. At the top, the Cisco logo and "NX-API Sandbox" are visible, along with navigation links for "Quick Start", "Yang Documentation", "Yang Models", and "Logout". The main area contains a text input field with the command "Logging level netstack 6". To the right, there are dropdown menus for "Method" (set to "RESTCONF (Yang)") and "Message format" (set to "json"). Below the input field, a "POST" dropdown is followed by a text box containing the URL "restconf/data/Cisco-NX-OS-device:System/". Three buttons, "Send", "Reset", and "Convert", are positioned to the right of the URL. Underneath, a "Request" tab is selected, showing a JSON payload:

```
{  "ipv4-items": {    "inst-items": {      "loggingLevel": "informational"    }  } }
```

 A "Copy" button is located to the right of the JSON. Below the request, a "Response" section is shown, which is currently empty, with a "Copy" button to its right.

Example:

For this example, the command is **logging level netstack 6** and the message format is xml:



The screenshot shows the Cisco NX-API Sandbox interface. At the top, there is a header with the Cisco logo, the text "NX-API Sandbox", and navigation links for "Quick Start", "Yang Documentation", "Yang Models", and "Logout". The main area is divided into two sections. The top section contains a text input field with the text "logging level **netstack** 6". To the right of this field are two dropdown menus: "Method:" set to "RESTCONF (Yang)" and "Message format:" set to "xml". Below the input field is a "POST" dropdown menu, a text input field containing "restconf/data/Cisco-NX-OS-device:System/", and three buttons: "Send" (orange), "Reset" (blue), and "Convert" (blue). Below this is a "Request" section with tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing an XML snippet:

```
<ipv4-items>
<inst-items>
  <loggingLevel>informational</loggingLevel>
</inst-items>
</ipv4-items>
```

 To the right of the XML is a green "Copy" button. Below the request is a "Response:" section, which is currently empty, with a green "Copy" button to its right.

Step 3 You can also convert the request into the following formats by clicking on the appropriate tab in the **Request** pane:

- Python
- Python3
- Java
- JavaScript
- Go-Lang

Note The Java-generated script does not work if you choose the PATCH option from the drop-down menu in the area above the Request tab. This is a known limitation with Java and is expected behavior.



PART IV

Model-Driven Programmability

- [Infrastructure Overview, on page 149](#)
- [Managing Components, on page 153](#)
- [OpenConfig YANG, on page 159](#)
- [NETCONF Agent, on page 167](#)
- [Converting CLI Commands to Network Configuration, on page 175](#)
- [RESTCONF Agent, on page 181](#)
- [gRPC Agent, on page 185](#)
- [Dynamic Logger, on page 197](#)
- [Model-Driven Telemetry, on page 205](#)



CHAPTER 14

Infrastructure Overview

This chapter contains the following topics:

- [About Model-Driven Programmability, on page 149](#)
- [About the Programmable Interface Infrastructure, on page 149](#)

About Model-Driven Programmability

The model-driven programmability of the NX-OS device allows you to automate the configuration and control of the device.

Data Modeling

Data modeling provides a programmatic and standards-based method of writing configurations to the network device, replacing the process of manual configuration. Data models are written in a standard, industry-defined language. Although configuration using a CLI may be more human-friendly, automating the configuration using data models results in better scalability.

The Cisco NX-OS device supports the YANG data modeling language. YANG is a data modeling language that is used to describe configuration and operational data, remote procedure calls, and notifications for network devices.

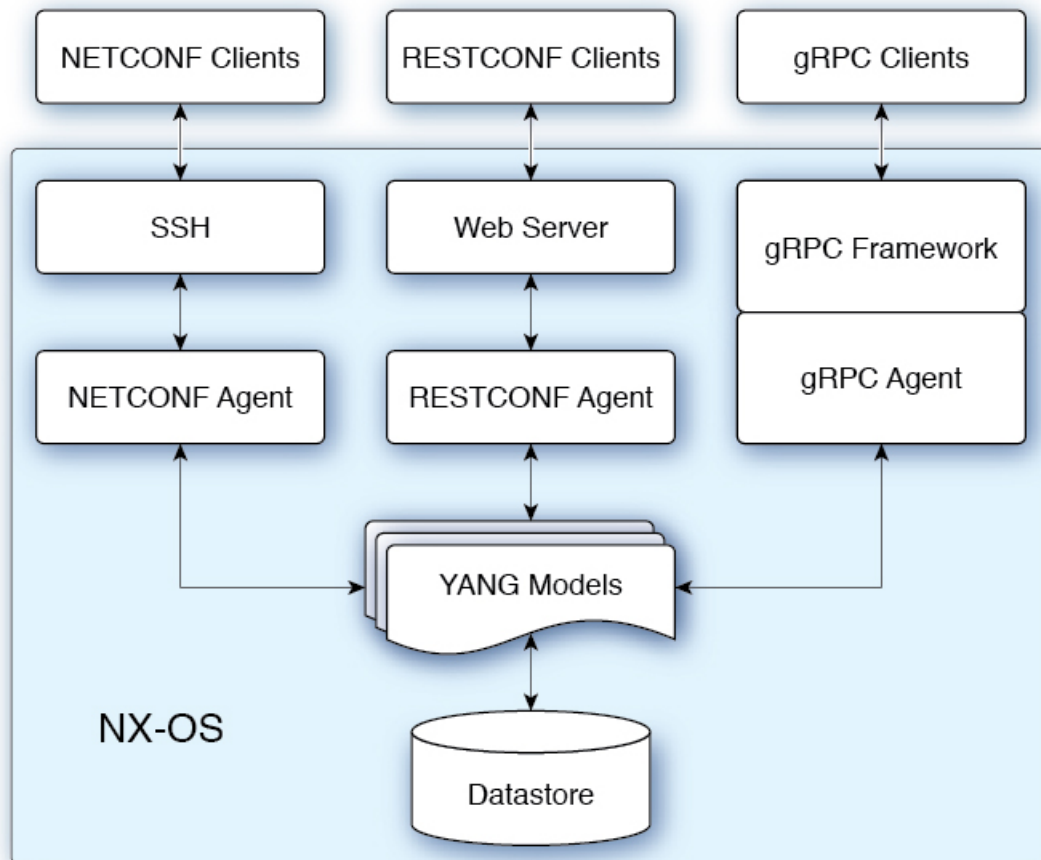
Programmable Interfaces

Three standards-based programmable interfaces are supported by NX-OS for operations on the data model:

- NETCONF
- RESTConf
- gRPC

About the Programmable Interface Infrastructure

This section provides a brief overview of the NX-OS Programmable Interface infrastructure.



When NX-OS receives a request whether through NETCONF, RESTConf, or gRPC, the request is converted into an abstract message object. That message object is distributed to the underlying model infrastructure based on the namespace in the request. Using the namespace, the appropriate model is selected and the request is passed to it for processing. The model infrastructure executes the request (read or write) on the device datastore. The results are returned to the agent of origin for response transmission back to the requesting client.

NX-OS Programmable Interface Agents

Agents provide an interface between the Device and clients. They specify the transport, the protocol, and the encoding of the communications with the Device. NX-OS Programmable Interfaces support three agents: NETCONF, RESTConf, and gRPC, each providing different interfaces for configuration management of the Device through YANG models.



Note Supported YANG models for each Cisco NX-OS release are provided at <https://devhub.cisco.com/artifactory/open-nxos-agents>.

Table 12: NX-OS Programmable Interface Agents

Agent	Transport	Protocol	Encoding
NETCONF	SSH		XML
RESTConf	HTTP	draft-ietf-netconf-restconf-10 ^[1]	XML or JSON
gRPC	HTTP	gRPC Protocol Spec ^[2]	Google Protobuf

The protocol specifications are described in the following documents:

- [1] RESTCONF Protocol draft-ietf-netconf-restconf-10 <https://tools.ietf.org/html/draft-ietf-netconf-restconf-10>
- [2] Cisco NX-OS gRPC Protocol Specification

Model Infrastructure

The Model Infrastructure takes requests that are received from the Agent, determines the namespace that is associated with the YANG model in the request, and selects the model component matching the namespace to process the request. When the selected model component completes request processing, the processing results are sent to the requesting Agent for transmission back to the client. The Model Infrastructure is also responsible for handling protocol initiation requests involving authentication, handshaking, and so on, as specified by the Agent protocol.

Device YANG Model

The Device Configuration is described in a YANG model that is called a Device Model. The Device Model is manifested in the Model Infrastructure as another model component with the Device namespace.

Common YANG Models

A Common Model is another kind of model component that contains within its elements, YANG Paths to the equivalent Device Model elements. These equivalent Device Model elements are used to read and write Device Model data in the Device YANG context.



CHAPTER 15

Managing Components

This chapter contains the following topics:

- [About the Component RPM Packages, on page 153](#)
- [Preparing for Installation, on page 155](#)
- [Downloading Components from the Cisco Artifactory, on page 156](#)
- [Installing RPM Packages, on page 156](#)

About the Component RPM Packages

NX-OS Programmable Interface Component RPM packages may be downloaded from the Cisco Artifactory. There are two types of component RPM packages that are needed:

- Base Components (required)
- Common Model Components (OpenConfig models must be explicitly downloaded and installed)

Base Components

The Base Components comprise the following required RPM packages:

- **mtx-infra** — Infrastructure
- **mtx-device** — Cisco native model

At least one of the following agent packages must be installed in order to have access to the modeled NX-OS interface:

- **mtx-netconf-agent** — NETCONF agent
- **mtx-restconf-agent** — RESTCONF agent
- **mtx-grpc-agent** — gRPC agent

Common Model Components

Common Model component RPMs support OpenConfig models. To use the OpenConfig models, you must download and install the OpenConfig RPMs. For convenience, there is a single combined package of all supported OpenConfig models, `mtx-openconfig-all`.

While the single combined package is recommended, an alternative is to download and install RPMs of selected models and their dependencies among the supported models listed in the following table. The `mtx-openconfig-all` RPM is not compatible with the individual model RPMs. You must uninstall the former before installing the latter, and you must uninstall the latter before installing the former.

Model Name	Model Rev	Model Ver	Package Name	Dependencies
openconfig-acl	2017-05-26	1.0.0	mtx-openconfig-acl	mtx-openconfig-interfaces
openconfig-bgp-policy	2017-07-30	4.0.1	mtx-openconfig-bgp-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-if-aggregate	2017-07-14	2.0.0	mtx-openconfig-if-aggregate	mtx-openconfig-if-ethernet mtx-openconfig-interfaces
openconfig-if-ethernet	2017-07-14	2.0.0	mtx-openconfig-if-ethernet	mtx-openconfig-interfaces
openconfig-if-ip	2016-05-26	1.0.2	mtx-openconfig-if-ip	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-if-ip-ext	2018-01-05	2.3.0	mtx-openconfig-if-ip-ext	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-if-ip mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-interfaces	2017-07-14	2.0.0	mtx-openconfig-interfaces	-
openconfig-network-instance	2017-08-24	0.8.1	mtx-openconfig-network-instance	mtx-openconfig-bgp-policy mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-routing-policy mtx-openconfig-vlan
openconfig-network-instance-policy	2017-02-15	0.1.0	mtx-openconfig-network-instance-policy	mtx-openconfig-routing-policy
openconfig-ospf-policy	2017-08-24	0.1.1	mtx-openconfig-ospf-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-platform	2018-01-16	0.8.0	mtx-openconfig-platform	-
openconfig-platform-linecard	2017-08-03	0.1.0	mtx-openconfig-platform-linecard	mtx-openconfig-platform

Model Name	Model Rev	Model Ver	Package Name	Dependencies
openconfig-platform-port	2018-01-20	0.3.0	mtx-openconfig-platform-port	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-platform-transceiver	2018-01-22	0.4.1	mtx-openconfig-platform-transceiver	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-relay-agent	2016-05-16	0.1.0	mtx-openconfig-relay-agent	mtx-openconfig-interfaces
openconfig-routing-policy	2016-05-12	2.0.1	mtx-openconfig-routing-policy	-
openconfig-spanning-tree	2017-07-14	0.2.0	mtx-openconfig-spanning-tree	mtx-openconfig-interfaces
openconfig-system	2017-09-18	0.3.0	mtx-openconfig-system	-
openconfig-vlan	2017-07-14	2.0.0	mtx-openconfig-vlan	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces

Preparing for Installation

This section contains installation preparation and other useful information for managing NX-OS Programmable Interface components.

Opening the Bash Shell on the Device

You perform RPM installation in the switch's Bash shell. Make sure that feature bash is configured on the device.

```
switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# feature bash-shell
Switch(config)# end
Switch# run bash sudo su
bash-4.2#
```

To return to the device CLI prompt from Bash, type **exit** or **Ctrl-D**.

Verify Device Readiness

You can use the following CLI **show** commands to confirm the readiness of the device before installation of an RPM.

- `show module`—Indicates whether all modules and LEMs are up.

```
Switch# show module
```

- `show system redundancy status`—Indicates whether the standby device is up and running and in HA mode. If a standby sync is in progress, the RPM installation may fail.

```
Switch# show system redundancy status
```

If the line cards have failed to come up, enter the `createrepo /rpms` command in the Bash shell.

```
bash-4.2# createrepo /rpms
```

Downloading Components from the Cisco Artifactory

You can download the NX-OS Programmable Interface Component RPMs from the Cisco Artifactory at the following URL. The RPMs are organized by NX-OS release-specific directories. Ensure that you are downloading the RPMs from the correct NX-OS release directory.

<https://devhub.cisco.com/artifactory/open-nxos-agents>

The NX-OS Programmable Interface Component RPMs adhere to the following naming convention:

`<package>-<version>-<NX-OS release>.<architecture>.rpm`

Select and download the desired NX-OS Programmable Interface Component RPM packages to the device for installation as described in the following sections.

Installing RPM Packages

Installing the Programmable Interface Base and Common Model Component RPM Packages

Before you begin

- From the Cisco Artifactory, download the following packages:
 - `mtx-infra`
 - `mtx-device`
 - `mtx-netconf-agent/mtx-restconf-agent/mtx-grpc-agent` (at least one)
 - `mtx-openconfig-all` (alternatively, selected individual models)
- Using the CLI commands in [Preparing for Installation, on page 155](#), confirm that all line cards in the Active and Standby devices are up and ready.

Procedure

- Step 1** Copy the downloaded RPMs to the device.

Example:

```
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
```

Step 2 From the Bash shell, install the RPMs.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

Step 3 From the Bash shell, verify the installation.

Example:

```
bash-4.2# yum list installed | grep mtx
```



CHAPTER 16

OpenConfig YANG

This chapter contains the following topics:

- [About OpenConfig YANG, on page 159](#)
- [Guidelines and Limitations for OpenConfig YANG, on page 159](#)
- [Understanding Deletion of BGP Routing Instance, on page 164](#)

About OpenConfig YANG

OpenConfig YANG supports modern networking principles, such as declarative configuration and model-driven management and operations. OpenConfig provides vendor-neutral data models for configuration and monitoring of the network, as well as helping with moving from a pull model to a push model, with subscriptions and event update streaming.

The Cisco Nexus 3400-S switches support a broad range of functional areas, including BGP, OSPF, Interface L2 and L3, VRFs, VLANs, and TACACS.

For more information about OpenConfig YANG, see <https://developer.cisco.com/docs/openconfig-yang/>.

For the OpenConfig models, see <https://github.com/YangModels/yang/tree/master/vendor/cisco/nx/>. OpenConfig YANG models are grouped by NX-OS release, so when the NX-OS release number changes, the last digits in the URL will change.

Guidelines and Limitations for OpenConfig YANG

OpenConfig YANG has the following guidelines and limitations:

- The following OpenConfig YANG limitations exist for OC-BGP-POLICY:
 - Action type is always permit for `community-set` and `as-path-set`, which applies to the following containers:
 - `/bgp-defined-sets/community-sets/community-set/`
 - `/bgp-defined-sets/as-path-sets/as-path-set/`

In OpenConfig YANG, there is no action type concept as there is in the CLI for `community-set` and `as-path-set`. Therefore, the action type is always permit for `community-set` and `as-path-set`.

- The following OpenConfig YANG limitation applies to this container:
/bgp-defined-sets/community-sets/community-set/

In the CLI, `community-list` can have two different types: standard and expanded. However, in the OpenConfig YANG model, `community-set-name` has no such differentiation.

When you create the `community-set-name` through OpenConfig YANG, the following things happen internally:

- The `_std` suffix is appended after `community-set-name` if `community-member` is in the standard form (AS:NN).
- The `_exp` suffix is appended after `community-set-name` if `community-member` is in the expanded form (regex):

```
<community-set>
  <community-set-name>oc_commsetld</community-set-name>
  <config>
    <community-set-name>oc_commsetld</community-set-name>
    <community-member>0:1</community-member>
    <community-member>_1_</community-member>
  </config>
</community-set>
```

The preceding OpenConfig YANG configuration is mapped to the following CLI:

```
ip community-list expanded oc_commsetld_exp seq 5 permit "_1_"
ip community-list standard oc_commsetld_std seq 5 permit 0:1
```

- The following OpenConfig YANG limitation applies to this container:
/bgp-conditions/match-community-set/config/community-set/

OpenConfig YANG can only map to one `community-set`, while the CLI can match to multiple instances of the `community-set`:

- In the CLI:

```
ip community-list standard 1-1 seq 1 permit 1:1
ip community-list standard 1-2 seq 1 permit 1:2
ip community-list standard 1-3 seq 1 permit 1:3
route-map To_LC permit 10
match community 1-1 1-2 1-3
```

- The corresponding OpenConfig YANG payload follows:

```
<config>
  <routing-policy xmlns="http://openconfig.net/yang/routing-policy">
    <defined-sets>
      <bgp-defined-sets xmlns="http://openconfig.net/yang/bgp-policy">
        <community-sets>
          <community-set>
            <community-set-name>cs</community-set-name>
            <config>
              <community-set-name>cs</community-set-name>
              <community-member>1:1</community-member>
              <community-member>1:2</community-member>
              <community-member>1:3</community-member>
            </config>
          </community-set>
        </community-sets>
      </bgp-defined-sets>
```



```

</defined-sets>
<policy-definitions>
  <policy-definition>
    <name>To_LC</name>
    <statements>
      <statement>
        <name>10</name>
        <conditions>
          <bgp-conditions xmlns="http://openconfig.net/yang/bgp-policy">
            <match-community-set>
              <config>
                <community-set>cs</community-set>
              </config>
            </match-community-set>
          </bgp-conditions>
        </conditions>
      </statement>
    </statements>
  </policy-definition>
</policy-definitions>
</routing-policy>
</config>

```

As a workaround, create one community with multiple statements through OpenConfig YANG:

```

ip community-list standard cs_std seq 5 permit 1:1
ip community-list standard cs_std seq 10 permit 1:2
ip community-list standard cs_std seq 15 permit 1:3
route-map To_LC permit 10
match community cs_std

```

- The following OpenConfig YANG limitation applies to this container:
/bgp-conditions/state/next-hop-in

In OpenConfig YANG, the `next-hop-in` type is an IP address, but in the CLI, it is an IP prefix.

While creating the `next-hop-in` through OpenConfig YANG, the IP address is converted to a "/32" mask prefix in the CLI configuration. For example:

- Following is an example of `next-hop-in` in the OpenConfig YANG payload:

```

<policy-definition>
  <name>sc0</name>
  <statements>
    <statement>
      <name>5</name>
      <conditions>
        <bgp-conditions xmlns="http://openconfig.net/yang/bgp-policy">
          <config>
            <next-hop-in>2.3.4.5</next-hop-in>
          </config>
        </bgp-conditions>
      </conditions>
    </statement>
  </statements>
</policy-definition>

```

- Following is an example of the same information in the CLI:

```

ip prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5 seq 5 permit 2.3.4.5/32
route-map sc0 permit 5

```

```
match ip next-hop prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5
```

- The following NX-OS limitations exist for OC-BGP-POLICY:
 - /bgp-actions/set-community/config/method enum "REFERENCE" is not supported.
 - enum "SELF", which is supported in the OpenConfig YANG model for /bgp-actions/config/set-next-hop, is not supported.
- For OC-BGP-POLICY, /bgp-conditions/match-community-set/config/community-set get mapped only to match community <community-set>_std, so only standard community is supported. Match to expanded community set is not supported.
- There is a limitation in replacing match-tag-set because defined sets for tag-sets are not currently implemented.

Currently, replacing match-tag-set appends the values. To replace match-tag-set, delete it, then create it again.

- The following guidelines and limitations apply to OSPF OpenConfig YANG:
 - If you configure and remove an area configuration in OSPF, the deleted areas (stale entries) are still shown in DME. Those stale area entries are shown in the GETCONFIG/GET output in OpenConfig YANG.
 - Only one area is supported in OpenConfig YANG in the OSPF policy match ospf-area configuration. In the CLI, you can configure to match multiple areas, such as match ospf-area 100 101. However, in OpenConfig YANG, you can configure only one area (for example, match ospf-area 100).
 - The area virtual-link and area interface configurations payload cannot go under the same area list. Split the area container payload as Virtual link area and interface area in the same payload.
 - The MD5 authentication string cannot be configured in OSPF OpenConfig YANG.

In the OSPF model, Authentication-type is defined for the Authentication:

```
leaf authentication-type {
  type string;
  description
    "The type of authentication that should be used on this
    interface";
}
```

OSPF OpenConfig YANG does not support an option for authentication password.

- The OSPF area authentication configuration is not supported. For example, area 0.0.0.200 authentication message-digest cannot be configured from OpenConfig YANG.
- The OSPF/BGP instance configuration that falls under default VRF (for example, router ospf 1/router bgp 1) will not get deleted when you delete the Protocols container with the default network instance.
- The following are guidelines and limitations for VLAN configuration between the OpenConfig payload and the Cisco Nexus 3400-S interfaces:

- When you attempt to simultaneously configure a trunk-mode interface and trunk VLANs in the same OpenConfig payload, the configuration does not complete successfully. However, when you split the payload so that the trunk-mode interface is sent first, then the trunk VLANs are sent, the configuration completes successfully.

On Cisco NX-OS interfaces the default interface mode is access. To implement any trunk-related configurations, you must first change the interface mode to trunk, then configure the trunk VLAN ranges. Do these configurations in separate payloads.

The following examples show the separate payloads for the configuring trunk mode and VLAN ranges.

Example 1, payload configuring the interface to trunk mode.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="http://openconfig.net/yang/interfaces">
        <interface>
          <name>eth1/47</name>
          <subinterfaces>
            <subinterface>
              <index>0</index>
              <config>
                <index>0</index>
              </config>
            </subinterface>
          </subinterfaces>
          <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
            <switched-vlan xmlns="http://openconfig.net/yang/vlan">
              <config>
                <interface-mode>TRUNK</interface-mode>
              </config>
            </switched-vlan>
          </ethernet>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

Example 2, payload configuring the VLAN ranges.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="http://openconfig.net/yang/interfaces">
        <interface>
          <name>eth1/47</name>
          <subinterfaces>
            <subinterface>
              <index>0</index>
              <config>
                <index>0</index>
              </config>
            </subinterface>
          </subinterfaces>
          <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
```

```

    <switched-vlan xmlns="http://openconfig.net/yang/vlan">
      <config>
        <native-vlan>999</native-vlan>
        <trunk-vlans xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete">1..4094</trunk-vlans>
          <trunk-vlans>401</trunk-vlans>
          <trunk-vlans>999</trunk-vlans>
        </config>
      </switched-vlan>
    </ethernet>
  </interface>
</interfaces>
</config>
</edit-config>
</rpc>

```

- Because of the design of OpenConfig YANG, when you configure VLANs, there must be no overlap between the VLANs in the payload and the VLANs already configured on an interface. If an overlap exists, the configuration through OpenConfig is not successful. Make sure that the VLANs configured on an interface are different from the VLANs in the OpenConfig payload. Pay particular attention to the starting and ending VLANs in a range.

Understanding Deletion of BGP Routing Instance

With OpenConfig YANG network-instance (OCNI), when attempting to delete only the BGP configuration of the default VRF instead of deleting the entire BGP routing instance, BGP information might not be deleted at the protocols/BGP level. In this situation, when the delete is at the protocols or BGP level with the autonomous system number in the payload, only the configuration of the default VRF is deleted instead of removing the entire BGP routing instance.

Following is an example payload that would be used to delete the configuration under the default VRF in BGP.

```

<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <network-instances xmlns="http://openconfig.net/yang/network-instance">
        <network-instance>
          <name>default</name>
          <protocols>
            <protocol>
              <identifier>BGP</identifier>
              <name>bgp</name>
              <bgp xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="delete">
                <global>
                  <config>
                    <as>100</as>
                  </config>
                </global>
              </bgp>
            </protocol>
          </protocols>
        </network-instance>
      </network-instances>
    </config>
  </edit-config>
</rpc>

```

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

Expected Behavior: The BGP routing instance itself should be deleted, which is the equivalent to **no router bgp 100**.

Actual Behavior: Only the BGP configuration under the default VRF is deleted. No equivalent single CLI configuration exists.

Following is the running configuration before the delete operation:

```
router bgp 100
  router-id 1.2.3.4
  address-family ipv4 unicast
  vrf abc
    address-family ipv4 unicast
    maximum-paths 2
```

And following is the running configuration after the delete operation:

```
router bgp 100
  vrf abc
    address-family ipv4 unicast
    maximum-paths 2
```




CHAPTER 17

NETCONF Agent

This chapter contains the following topics:

- [About the NETCONF Agent, on page 167](#)
- [Guidelines and Limitations, on page 169](#)
- [Configuring the NETCONF Agent, on page 169](#)
- [Using the NETCONF Agent, on page 170](#)
- [Troubleshooting the NETCONF Agent, on page 174](#)

About the NETCONF Agent

The Cisco NX-OS NETCONF Agent is a client-facing interface. The agent provides secure transport for the client requests and server responses in the form of a YANG model, encoded in XML.

The NETCONF Agent supports a candidate configuration feature. The Candidate configuration datastore temporarily holds candidate configuration and any changes you make without changing the running configuration. You can then choose when to update the configuration of the device with the candidate configuration when you commit and confirm the candidate configuration.

If you do not confirm the changes, exit from a nonpersistent NETCONF client session, or choose to cancel the commit after you commit the change, a system timer then times out and rolls back the changes.

If you initiate a confirmed-commit operation with a persistent token, the NETCONF client session becomes a persistent process. In a persistent process, exiting the NETCONF client session does not call an automatic roll-back. Also, the changes cannot be rolled back without the matching persistent token.

Cisco NX-OS NETCONF supports the following configuration capabilities:

- Writable-Running Capability

```
urn:ietf:params:netconf:capability:writable-running:1.0
```

- Rollback-on-error Capability

```
urn:ietf:params:netconf:capability:rollback-on-error:1.0
```

- Candidate Configuration Capability

```
urn:ietf:params:netconf:capability:candidate:1.0
```

- Validation Capability

```
urn:ietf:params:netconf:capability:validate:1.1
```


- discard-changes



Note The delete-config operation is not allowed.

Guidelines and Limitations

The NETCONF Agent has the following guideline and limitation:

- NETCONF does not support enhanced Role-Based Access Control (RBAC) as specified in RFC 6536. Only users with a "network-admin" role are granted access to the NETCONF agent.
- NETCONF requires RPM to be installed on port 830.

Configuring the NETCONF Agent

The NETCONF Agent supports the following optional configuration parameters under the [netconf] section in the configuration file (/etc/mtx.conf).

Parameter	Description
idle_timeout	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The default value is 5 minutes. A value of 0 disables timeout.
limit	(Optional) Specifies the number of maximum simultaneous client sessions. The default value is 5 sessions. The range is 1 to 50.

The following is an example of the [netconf] section in the configuration file:

```
[netconf]
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

For the modified configuration file to take effect, you must restart the NETCONF Agent using the CLI command **[no] feature netconf** to disable and reenab.

Using the NETCONF Agent

General Commands

The NETCONF Agent is enabled or disabled by the command `[no] feature netconf`.

General Control Commands

The available control commands for the NETCONF agent are:

```
netconfctl { status | start | restart | reload | stop }
```

Viewing the Agent Status

```
bash-4.2# netconfctl status
xosdsd is stopped
netconf is stopped
```

Starting the Agent

```
bash-4.2# netconfctl start
Starting Netconf Agent: [OK]
```

Initializing the Candidate Configuration Datastore

The candidate configuration can only be initialized with the contents of the running configuration. To initialize the candidate configuration datastore, send a Copy-Config request using SSH, with candidate as the target and running as the source.

Performing Read and Write on the Candidate Configuration

To read from the candidate configuration, send a Get-Config request with SSH, using candidate as the source.

To write to the contents of the candidate configuration, send an Edit-Config request with SSH, using candidate as the target.

NETCONF Candidate Configuration Workflow

The candidate configuration workflow is as follows:

- Edit the candidate configuration file.
- Validate the candidate configuration.
- Commit the changes to the running configuration.

Example: An SSH Session

This example shows initiating a session using the SSH client and sending an Edit-Config and GET request using the SSH Client.

```

client-host % ssh -s admin@172.19.193.152 -p 830 netconf
User Access Verification
Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello>
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>
    <capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2017-04-06&
  </capabilities>
  <module>cisco-nx-os-device&deviations=cisco-nx-os-device-deviations</module>
  </capabilities>
  <session-id>1912037714</session-id>
</hello>
]]>]]<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
#794
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bgp-items>
          <inst-items>
            <dom-items>
              <Dom-list>
                <name>default</name>
                <rtrId>2.2.2.2</rtrId>
              </Dom-list>
            </dom-items>
          </inst-items>
        </bgp-items>
      </System>
    </config>
  </edit-config>
</rpc>
##

#190
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

##

#511
<rpc message-id="109"

```

```

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
  <source>
    <running/>
  </source>
  <filter type="subtree">
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <bgp-items>
        <inst-items>
          <dom-items>
            <Dom-list/>
          </dom-items>
        </inst-items>
      </bgp-items>
    </System>
  </filter>
</get-config>
</rpc>
##

#996
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="109"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <System>
      <bgp-items>
        <inst-items>
          <dom-items>
            <Dom-list>
              <name>default</name>
              <always>disabled</always>
              <bestPathIntvl>300</bestPathIntvl>
              <holdIntvl>180</holdIntvl>
              <kaIntvl>60</kaIntvl>
              <maxAsLimit>0</maxAsLimit>
              <pfxPeerTimeout>30</pfxPeerTimeout>
              <pfxPeerWaitTime>90</pfxPeerWaitTime>
              <reConnIntvl>60</reConnIntvl>
              <rtrId>2.2.2.2</rtrId>
            </Dom-list>
          </dom-items>
        </inst-items>
      </bgp-items>
    </System>
  </data>
</rpc-reply>
##

```

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

- create
- merge
- delete

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

```

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

```

Error Messages

If a request results in an error, the response payload includes the error.

Errors Defined by Cisco

The following are the errors that are defined by Cisco.

Error defined by Cisco	Description
unknown-error-cond	Unknown error encountered.
n-y-i	The requested operation is not supported (not-yet-implemented)
namespace-not-found	Error in the request payload.
namespace-already-exists	Error in the request payload.
object-not-found	Error in the request payload.
object-not-container	Error in the request payload.
object-not-property	Error in the request payload.
no-property-in-object	Error in the request payload.
invalid-dn	Internal error.
invalid-arg	Internal error.
already-exists	Error in the request payload.
container-not-found	Error in the request payload
container-already-exists	Error in the request payload.
property-not-found	Error in the request payload.
property-already-exists	Error in the request payload.
malformed	Error in the request payload.
alloc-failed	Internal error.
sigint	Internal error.

not-initialized	Internal error.
inappropriate	Internal error.

The following is an example of a NETCONF error response payload that reports an invalid IP address value:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="320" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>Protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>Error</error-severity>
    <error-message xml:lang="en">Property Merge (set property) Failed: operation-failed
value=500.500.500.500</error-message>
    <error-path>/config/System/bgp-items/inst-items/dom-items/Dom-list/rtrId</error-path>
  </rpc-error>
</rpc-reply>
```

Troubleshooting the NETCONF Agent

Troubleshooting Connectivity

- From a client system, ping the management port of the switch to verify that the switch is reachable.
- In the Bash shell of the switch, execute the **service netconf status** command to check the agent status.
- Check whether the RSA host key for SSH is outdated. If so, remove the RSA host key entry of the switch from the `~/.ssh/known_hosts` file on the client host. This example shows the message that is received when the host key is outdated:

```
client-host % ssh -s admin@192.0.20.111 -p 830 netconf
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@ @ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-
the-middle attack)! It is also possible that the RSA host
key has just been changed. The fingerprint for the RSA key
sent by the remote host is
82:3d:49:5c:1b:08:4c:8e:19:94:a8:1f:32:8d:1e:dd. Please
contact your system administrator. Add correct host key
in /users/myuser/.ssh/known_hosts to get rid of this
message. Offending key in
/users/myuser/.ssh/known_hosts:304 Password
authentication is disabled to avoid man-in-the-middle
attacks. Keyboard-interactive authentication is disabled
to avoid man-in-the-middle attacks. User Access
Verification Permission denied
(publickey,password,keyboard-interactive).
client-host %
```



CHAPTER 18

Converting CLI Commands to Network Configuration

This chapter contains the following contents:

- [Information About XMLIN, on page 175](#)
- [Licensing Requirements for XMLIN, on page 175](#)
- [Installing and Using the XMLIN Tool, on page 176](#)
- [Converting Show Command Output to XML, on page 176](#)
- [Configuration Examples for XMLIN, on page 177](#)

Information About XMLIN

The XMLIN tool converts CLI commands to the Network Configuration (NETCONF) protocol format. NETCONF is a network management protocol that provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses XML-based encoding for configuration data and protocol messages. The NX-OS implementation of the NETCONF protocol supports the following protocol operations: `<get>`, `<edit-config>`, `<close-session>`, `<kill-session>`, and `<exec-command>`.

The XMLIN tool converts show, EXEC, and configuration commands to corresponding NETCONF `<get>`, `<exec-command>`, and `<edit-config>` requests. You can enter multiple configuration commands into a single NETCONF `<edit-config>` instance.

The XMLIN tool also converts the output of show commands to XML format.

Licensing Requirements for XMLIN

Table 13: XMLIN Licensing Requirements

Product	License Requirement
Cisco NX-OS	XMLIN requires no license. Any feature not included in a license package is bundled with the Cisco NX-OS system images and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the <i>Cisco NX-OS Licensing Guide</i> .

Installing and Using the XMLIN Tool

You can install the XMLIN tool and then use it to convert configuration commands to NETCONF format.

Before you begin

The XMLIN tool can generate NETCONF instances of commands even if the corresponding feature sets or required hardware capabilities are not available on the device. But, you might still need to install some feature sets before entering the **xmlin** command.

Procedure

	Command or Action	Purpose
Step 1	switch# xmlin	
Step 2	switch(xmlin)# configure terminal	Enters global configuration mode.
Step 3	Configuration commands	Converts configuration commands to NETCONF format.
Step 4	(Optional) switch(config)(xmlin)# end	Generates the corresponding <edit-config> request. Note Enter the end command to finish the current XML configuration before you generate an XML instance for a show command.
Step 5	(Optional) switch(config-if-verify)(xmlin)# show commands	Converts show commands to NETCONF format.
Step 6	(Optional) switch(config-if-verify)(xmlin)# exit	Returns to EXEC mode.

Converting Show Command Output to XML

You can convert the output of show commands to XML.

Before you begin

Make sure that all features for the commands you want to convert are installed and enabled on the device. Otherwise, the commands fail.

You can use the **terminal verify-only** command to verify that a feature is enabled without entering it on the device.

Make sure that all required hardware for the commands you want to convert are present on the device. Otherwise, the commands fail.

Make sure that the XMLIN tool is installed.

Procedure

	Command or Action	Purpose
Step 1	switch# <i>show-command</i> xmlin	Enters global configuration mode. Note You cannot use this command with configuration commands.

Configuration Examples for XMLIN

The following example shows how the XMLIN tool is installed on the device and used to convert a set of configuration commands to an <edit-config> instance.

```
switch# xmlin
*****
Loading the xmlin tool. Please be patient.
*****
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2019, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.

switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)(xmlin)# interface ethernet 2/1
% Success
switch(config-if-verify)(xmlin)# cdp enable
% Success
switch(config-if-verify)(xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec"
xmlns:ml="http://www.cisco.com/nxos:6.2.2.:configure__if-eth-base" message-id="1">
  <nf:edit-config>
    <nf:target>
      <nf:running/>
    </nf:target>
  <nf:config>
    <m:configure>
      <m:terminal>
        <interface>
          <__XML_PARAM__interface>
```

```

        <__XML__value>Ethernet2/1</__XML__value>
        <ml:cdp>
            <ml:enable/>
        </ml:cdp>
    </__XML__PARAM__interface>
</interface>
</m:terminal>
</m:configure>
</nf:config>
</nf:edit-config>
</nf:rpc>
]]>]]>

```

The following example shows how to enter the **end** command to finish the current XML configuration before you generate an XML instance for a **show** command.

```

switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config) (xmlin)# interface ethernet 2/1
switch(config-if-verify) (xmlin)# show interface ethernet 2/1
*****
Please type "end" to finish and output the current XML document before building a new one.
*****
% Command not successful

switch(config-if-verify) (xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec" message-id="1">
  <nf:edit-config>
    <nf:target>
      <nf:running/>
    </nf:target>
    <nf:config>
      <m:configure>
        <m:terminal>
          <interface>
            <__XML__PARAM__interface>
              <__XML__value>Ethernet2/1</__XML__value>
            </__XML__PARAM__interface>
          </interface>
        </m:terminal>
      </m:configure>
    </nf:config>
  </nf:edit-config>
</nf:rpc>
]]>]]>

switch(xmlin)# show interface ethernet 2/1
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager" message-id="1">
  <nf:get>
    <nf:filter type="subtree">
      <show>
        <interface>
          <__XML__PARAM__ifeth>
            <__XML__value>Ethernet2/1</__XML__value>
          </__XML__PARAM__ifeth>
        </interface>
      </show>
    </nf:filter>

```

```
</nf:get>
</nf:rpc>
]]>]]>
switch(xmlin)# exit
switch#
```

The following example shows how you can convert the output of the **show interface brief** command to XML.

```
switch# show interface brief | xmlin
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager"

message-id="1">
  <nf:get>
    <nf:filter type="subtree">
      <show>
        <interface>
          <brief/>
        </interface>
      </show>
    </nf:filter>
  </nf:get>
</nf:rpc>
]]>]]>
```




CHAPTER 19

RESTCONF Agent

This chapter contains the following topics:

- [About the RESTConf Agent, on page 181](#)
- [Guidelines and Limitations, on page 182](#)
- [Configuring the RESTConf Agent, on page 182](#)
- [Using the RESTConf Agent, on page 182](#)
- [Troubleshooting the RESTConf Agent, on page 183](#)

About the RESTConf Agent

Cisco NX-OS RESTConf is an HTTP -based protocol for configuring data that YANG version 1 defines, using datastores in NETCONF.

NETCONF defines configuration datastores and a set of Create, Retrieve, Update, and Delete (CRUD) operations that can access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and event notifications.

Cisco NX-OS RESTConf uses HTTP operations to provide CRUD operations on a conceptual datastore containing YANG-defined data. This data is compatible with a server which implements NETCONF datastores.

The RESTCONF protocol supports both XML and JSON payload encodings. User authentication happens through the HTTP Basic Authentication.

The following table shows the Protocol operations that the Cisco NX-OS RESTConf Agent supports:

RESTConf	NETCONF Equivalent
OPTIONS	NETCONF: none
HEAD	NETCONF: none
GET	NETCONF: <get-config>, <get>
POST	NETCONF: <edit-config> (operation="create")
PUT	NETCONF: <edit-config> (operation="create/replace")
PATCH	NETCONF: <edit-config> (operation="merge")
DELETE	NETCONF: <edit-config> (operation="delete")

Guidelines and Limitations

The RESTCONF Agent has the following guideline and limitation:

- Cisco NX-OS RESTCONF is based on an RFC draft entitled RESTCONF Protocol draft-ietf-netconf-restconf-10. See the <https://tools.ietf.org/html/draft-ietf-netconf-restconf-10> document.
- RESTCONF does not support enhanced Role-Based Access Control (RBAC) as specified in RFC 6536. Only the users with a "network-admin" role are granted access to the RESTCONF agent.

Configuring the RESTConf Agent

The RESTConf Agent does not require any configuration in the configuration file (/etc/mtx.conf).

Using the RESTConf Agent

General Commands

- You can enable or disable the RESTCONF Agent **[no] feature restconf** command.
- Configure the following commands to enable HTTP or HTTPS access:
 - **feature nxapi**
 - **nxapi http port 80**
 - **nxapi https port 443**

General Control Commands

The available control commands for the RESTConf agent are:

```
restconfctl { status | start | restart | reload | stop }
```

Viewing the Agent Status

```
bash-4.2# restconfctl status
xosdsd is stopped
restconfctl is stopped
```

Starting the Agent

```
bash-4.2# restconfctl start
Starting Restconf Agent: [OK]
```

Sending a POST Request to the Server Using Curl

```
client-host % curl -X POST -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Content-Type:
application/yang.data+xml" -d '<always>enabled</always><rtrId>2.2.2.2</rtrId>'
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list=default"
-i

HTTP/1.1 201 Created
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:25:31 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500853169574134
Status: 201 Created
Location: /System/bgp-items/inst-items/dom-items/Dom-list=default/always/rtrId/
```

Sending a GET Request to the Server Using Curl

```
client-host % curl -X GET -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Accept:
application/yang.data+xml"
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list?content=config"
-i

HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:26:03 GMT
Content-Type: application/yang.data+xml
Content-Length: 395
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500856185650327
Status: 200 OK

    <Dom-list>
      <name>default</name>
      <always>enabled</always>
      <bestPathIntvl>300</bestPathIntvl>
      <holdIntvl>180</holdIntvl>
      <kaIntvl>60</kaIntvl>
      <maxAsLimit>0</maxAsLimit>
      <pfxPeerTimeout>30</pfxPeerTimeout>
      <pfxPeerWaitTime>90</pfxPeerWaitTime>
      <reConnIntvl>60</reConnIntvl>
      <rtrId>2.2.2.2</rtrId>
    </Dom-list>
client-host %
```

Troubleshooting the RESTConf Agent

Troubleshooting Connectivity

- Enable the web server by issuing the **feature nxapi** command.
- Ensure that the **nxapi http port 80** command is configured to open up the port for HTTP
- Ensure that the **nxapi https port 443** command is configured to open up the port for HTTPS.

- Ping the management port of the switch to verify that the switch is reachable.
- In the Bash shell of the switch, execute the **service restconf status** command to check the agent status.



CHAPTER 20

gRPC Agent

This chapter contains the following topics:

- [About the gRPC Agent, on page 185](#)
- [Guidelines and Limitations, on page 186](#)
- [Configuring the gRPC Agent, on page 187](#)
- [Using the gRPC Agent, on page 188](#)
- [Troubleshooting the gRPC Agent, on page 189](#)
- [gRPC Protobuf File, on page 189](#)

About the gRPC Agent

The Cisco NX-OS gRPC protocol defines a mechanism through which a network device can be managed and its configuration data can be retrieved and installed. The protocol exposes a complete and formal Application Programming Interface (API) that clients can use to manage device configurations.

The Cisco NX-OS gRPC protocol uses a remote procedure call (RPC) paradigm where an external client manipulates device configurations utilizing Google Protocol Buffer (GPB)-defined API calls along with their service-specific arguments. These GPB-defined APIs transparently cause an RPC call to the device that return replies in the same GPB-defined API context.

The gRPC Agent provides a secure transport through TLS and user authentication and authorization through AAA.

The functional objective of the Cisco NX-OS gRPC protocol is to mirror that provided by NETCONF, particularly in terms of both stateless and stateful configuration manipulation for maximum operational flexibility.

The Cisco NX-OS gRPC Agent supports the following protocol operations:

- Get
- GetConfig
- GetOper
- EditConfig
- StartSession
- CloseSession

- KillSession

The gRPC Agent supports two types of operations:

- **Stateless operations** are performed entirely within a single message without creating a session.
- **Stateful operations** are performed using multiple messages. The following is the sequence of operations that are performed:
 1. Start the session. This action acquires a unique session ID.
 2. Perform session tasks using the session ID.
 3. Close the session. This action invalidates the session ID.

The following are the supported operations. See the Appendix for their RPC definitions in the .proto file that is exported by the gRPC Agent.

Operation	Description
StartSession	Starts a new session between the client and server and acquires a unique session ID.
EditConfig	Writes the specified YANG data subset to the target datastore.
GetConfig	Retrieves the specified YANG configuration data subset from the source datastore.
GetOper	Retrieves the specified YANG operational data from the source datastore.
Get	Retrieves the specified YANG configuration and operational data from the source datastore.
KillSession	Forces the termination of a session.
CloseSession	Requests graceful termination of a session.

GetConfig, GetOper, and Get are stateless operations so don't require a session ID.

EditConfig can be either stateless or stateful. For a stateless operation, specify the SessionID as 0. For a stateful operation, a valid (nonzero) SessionID is required.

The gRPC Agent supports timeout for the sessions. The idle timeout for sessions can be configured on the device, after which idle sessions are closed and deleted.

Guidelines and Limitations

The gRPC Agent has the following guideline and limitation:

- gRPC does not support enhanced Role-Based Access Control (RBAC) as specified in RFC 6536. Only users with a "network-admin" role are granted access to the gRPC agent.

Configuring the gRPC Agent

The gRPC Agent supports the following configuration parameters under the `[grpc]` section of the configuration file (`/opt/mtx/etc/grpc.key`).

Parameter	Description
idle_timeout	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The default timeout is 5 minutes. A value of 0 disables timeout.
limit	(Optional) Specifies the number of maximum simultaneous client sessions. The default limit is 5 sessions. The range is from 1 through 50.
lport	(Optional) Specifies the port number on which the gRPC Agent listens. The default port is 50051.
key	Specifies the key file location for TLS authentication. The default location is <code>/opt/mtx/etc/grpc.key</code>
cert	Specifies the certificate file location for TLS authentication. The default location is <code>/opt/mtx/etc/grpc.key</code>
security	Specifies the type of secure connection. Valid choices are: <ul style="list-style-type: none"> • TLS for TLS • NONE for an unsecure connection

The following is an example of the `[grpc]` section in the configuration file:

```
[grpc]
mtxadapter=/opt/mtx/lib/libmtxadaptergrpc.1.0.1.so
idle_timeout=10
limit=1
lport=50051
security=TLS
cert=/etc/grpc.pem
key=/etc/grpc.pem
```

For the modified configuration file to take effect, you must restart the gRPC Agent using the CLI command `[no] feature grpc` to disable and reenale.

Using the gRPC Agent

General Commands

You can enable or disable the gRPC Agent by issuing the `[no] feature grpc` command.

Example: A Basic Yang Path in JSON Format

```
client-host % cat payload.json
{
  "namespace": "http://cisco.com/ns/yang/cisco-nx-os-device",
  "System": {
    "bgp-items": {
      "inst-items": {
        "dom-items": {
          "Dom-list": {
            "name": "default",
            "rtrId": "7.7.7.7",
            "holdIntvl": "100"
          }
        }
      }
    }
  }
}
```



Note The JSON structure has been pretty-formatted here for readability.

Sending an EditConfig Request to the Server

```
client-host % ./grpc_client -username=admin -password=cisco -operation=EditConfig
-e_oper=Merge -def_op=Merge -err_op=stop-on-error -infile=payload.json -reqid=1
-source=running -tls=true -serverAdd=192.0.20.123 -lport=50051
```

```
#####
Starting the client service
#####
TLS set true for client requestslems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending EditConfig request to the server
sessionid is
0
reqid:1
{"rpc-reply":{"ok":""}}
```

Sending a GetConfig Request to the Server

```
client-host % ./grpc_client -username=admin -password=cisco -operation=GetConfig
-infile=payload.json -reqid=1 -source=running -tls=true -serverAdd=192.0.20.123 -lport=50051
```

```
#####
Starting the client service
#####
TLS set true for client requestslems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending GetConfig request to the server
in get config
Got the response from the server
#####
Yang Json is:
#####
{"rpc-reply":{"data":{"System":{"top-items":{"inst-items":{"don-items":{"Don-List":{"name":"default","rtid":"7.7.7","holdIntvl":"100"}}}}}}}}
#####
client-host %
```

Troubleshooting the gRPC Agent

Troubleshooting Connectivity

- From a client system, verify that the agent is listening on the port. For example:

```
client-host % nc -z 192.0.20.222 50051
Connection to 192.0.20.222 50051 port [tcp/*] succeeded!
client-host % echo $?
0
client-host %
```

- In the Bash shell of the switch, execute the **service grpc status** command to check the agent status.

gRPC Protobuf File

The gRPC Agent exports the supported operations and data structures in the proto definition file at `/opt/mtx/etc/nxos_grpc.proto`. The file is included in the gRPC Agent RPM. The following shows the definitions:

```
// Copyright 2016, Cisco Systems Inc.
// All rights reserved.

syntax = "proto3";

package NXOSExtensibleManagabilityService;

// Service provided by Cisco NX-OS gRPC Agent
service gRPCConfigOper {

    // Retrieves the specified YANG configuration data subset from the
    // source datastore
    rpc GetConfig(GetConfigArgs) returns(stream GetConfigReply) {};

    // Retrieves the specified YANG operational data from the source datastore
```

```

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

// Retrieves the specified YANG configuration and operational data
// subset from the source datastore
rpc Get(GetArgs) returns(stream GetReply) {};

// Writes the specified YANG data subset to the target datastore
rpc EditConfig(EditConfigArgs) returns(EditConfigReply) {};

// Starts a new session between the client and server and acquires a
// unique session ID
rpc StartSession(SessionArgs) returns(SessionReply) {};

// Requests graceful termination of a session
rpc CloseSession(CloseSessionArgs) returns (CloseSessionReply) {};

// Forces the termination of a session
rpc KillSession(KillArgs) returns(KillReply) {};

// Unsupported; reserved for future
rpc DeleteConfig(DeleteConfigArgs) returns(DeleteConfigReply) {};

// Unsupported; reserved for future
rpc CopyConfig(CopyConfigArgs) returns(CopyConfigReply) {};

// Unsupported; reserved for future
rpc Lock(LockArgs) returns(LockReply) {};

// Unsupported; reserved for future
rpc UnLock(UnLockArgs) returns(UnLockReply) {};

// Unsupported; reserved for future
rpc Commit(CommitArgs) returns(CommitReply) {};

// Unsupported; reserved for future
rpc Validate(ValidateArgs) returns(ValidateReply) {};

// Unsupported; reserved for future
rpc Abort(AbortArgs) returns(AbortReply) {};

}

message GetConfigArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;
}

message GetConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails

```

```
        string Errors = 3;
    }

message GetOperArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message GetOperReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message GetArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath=1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message GetReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message EditConfigArgs
{
    // JSON-encoded YANG data to be edited
    string YangPath = 1;

    // Specifies the operation to perform on teh configuration datastore with
    // the YangPath data. Possible values are:
    // create
    // merge
    // replace
    // delete
    // remove
    // If not specified, default value is "merge".
    string Operation = 2;

    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 3;
}
```

```
// (Optional) Specifies the request ID. Default value is 0.
int64 ReqID = 4;

// (Optional) Specifies the target datastore; only "running" is supported.
// Default is "running".
string Target = 5;

// Specifies the default operation on the given object while traversing
// the configuration tree.
// The following operations are possible:
//   merge:    merges the configuration data with the target datastore;
//             this is the default.
//   replace:  replaces the configuration data with the target datastore.
//   none:     target datastore is unaffected during the traversal until
//             the specified object is reached.
string DefOp = 6;

// Specifies the action to be performed in the event of an error during
// configuration. Possible values are:
//   stop
//   roll-back
//   continue
// Default is "roll-back".
string ErrorOp = 7;
}

message EditConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If EditConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message DeleteConfigArgs
{
    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message DeleteConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If DeleteConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}
```



```
message CopyConfigArgs
{
    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 4;
}

message CopyConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CopyConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message LockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID=2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message LockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Lock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message UnLockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the target datastore; only "running" is supported.
```

```
    // Default is "running".
    string Target = 3;
}

message UnLockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If UnLock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message SessionArgs
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;
}

message SessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;
    int64 SessionID = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CloseSessionArgs
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;

    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 2;
}

message CloseSessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CloseSession is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message KillArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    int64 SessionIDToKill = 2;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 3;
}
```

```
message KillReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Kill is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message ValidateArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message ValidateReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Validate is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CommitArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message CommitReply
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;

    // If Commit is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message AbortArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message AbortReply
```

```
{
  // (Optional) Specifies the request ID. Default value is 0.
  int64 ReqID = 1;

  // If Abort is successful, YangData contains a JSON-encoded "ok" response.
  string YangData = 2;

  // JSON-encoded error information when request fails
  string Errors = 3;
}
```



CHAPTER 21

Dynamic Logger

This chapter contains the following topics:

- [Prerequisites, on page 197](#)
- [Reference, on page 197](#)

Prerequisites

Before using dynamic logging, confirm that the following are on your switch:

- The `libmtxlogmgr*.so` library is installed `/opt/mtx/lib/`. The `libmtxlogmgr*.so` library is part of the `mtx_infra` RPM.
- The `mtx.conf` file that is located in `/etc/` contains:

```
[mtxlogger]
config=/opt/mtx/conf/mtxlogger.cfg
```

- The `mtxlogger.cfg` file is in `/opt/mtx/conf/`.

Reference

The configuration file has the following structure:

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false"></leaf>
      <leaf name="allActive" type="boolean" default="false"></leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$"></leaf>
      <container name="componentID">
        <leaf name="enabled" type="boolean" default="true"></leaf>
      </container>
      <container name="dateTime">
        <leaf name="enabled" type="boolean" default="true"></leaf>
        <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
      </container>
      <container name="fcn">
        <leaf name="enabled" type="boolean" default="true"></leaf>
        <leaf name="format" type="string">
```

```

default="$CLASS::$FCNNAME$($ARGSS)@$LINE$"></leaf>
</container>
</container>
<container name="dest">
  <container name="console">
    <leaf name="enabled" type="boolean" default="false"></leaf>
  </container>
  <container name="file">
    <leaf name="enabled" type="boolean" default="false"></leaf>
    <leaf name="name" type="string" default="mtx-internal.log"></leaf>
    <leaf name="location" type="string" default="./mtxlogs"></leaf>
  </container>
</container>
<leaf name="mbytes-rollover" type="uint32" default="10"></leaf>
<leaf name="hours-rollover" type="uint32" default="24"></leaf>
<leaf name="startup-rollover" type="boolean" default="false"></leaf>
  <leaf name="max-rollover-files" type="uint32" default="10"></leaf>
</container>
</container>
<list name="logitems" key="id">
  <listitem>
    <leaf name="id" type="string"></leaf>
  </listitem>
</list>
</container>
</container>
</config>

```

The `<list>` tag defines the log filters by `<componentID>`.

The following table describes some of the containers and their leaves.

Table 14: Container and Leaf Descriptions

Container	Container Description	Contained Containers	Contained Leaf and Description
logging	Contains all logging data types	format dest file Note Also contains list tag "logitems"	enabled: Boolean that determines whether logging is on or off. Default off. allActive: Boolean that activates all defined logging items for logging. Default off

Container	Container Description	Contained Containers	Contained Leaf and Description
format	Contains the log message format information	componentID dateTime type fcn	content: String listing data types included in log messages. Includes: <ul style="list-style-type: none"> • \$DATETIMES\$: Include date or time in the log message • \$COMPONENTID\$: Include component name in the log message. • \$TYPES\$: Includes message type ("", INFO, WARNING, ERROR) • \$SRCFILES\$: Includes the name of the source file. • \$SRCLINES\$: Include the line number of the source file • \$FCNINFOS\$ Include class::function name from the source file. • \$MSG\$: Include the actual log message text.
componentID	Name of the logged component.	NA	enabled: Boolean that determines if the log message includes the component ID. Default to "true." Value of "false" returns a "" string in the log message.

Container	Container Description	Contained Containers	Contained Leaf and Description
dateTime	Date or time of the log message	NA	<p>enabled: Boolean whether to include date or time information in the log message. Default is enabled.</p> <p>format: String of values to include in the log message. Format of %y%m%d.%H%M%S.</p>
dest	Holds destination logger's configuration settings.	<p>console: Destination console. Only one allowed.</p> <p>file: destination file. Multiple allowed.</p>	NA
console	Destination console	NA	enabled: Boolean that determines whether the console is enabled for logging. Default of "false."

Container	Container Description	Contained Containers	Contained Leaf and Description
file	Determines the settings of the destination file.	NA	<p>enabled: Boolean that determines whether the destination is enabled. Default is "false."</p> <p>name: String of the destination log file. Default of "mtx-internal.log"</p> <p>location: String of destination file path. Default at "./mtxlogs."</p> <p>mbytes-rollover: uint32 that determines the length of the log file before the system overwrites the oldest data. Default is 10 Mbytes.</p> <p>hours-rollover: uint32 that determines the length of the log file in terms of hours. Default is 24 hours.</p> <p>startup-rollover: Boolean that determines if the log file is rolled over upon agent start or restart. Default value of "false."</p> <p>max-rollover-files: uint32 that determines the maximum number of rollover files; deletes the oldest file when the max-rollover-files value exceeded. Default value of 10.</p>

Example

The following is the configuration file with the default installed configuration.

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">>true</leaf>
      <leaf name="allActive" type="boolean" default="false">>false</leaf>
    <container name="format">
      <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$">$DATETIME$ $COMPONENTID$ $TYPE$ $SRCFILE$ @ $SRCLINE$ $FCNINFO$: $MSG$</leaf>
```

```

    <container name="componentID">
      <leaf name="enabled" type="boolean" default="true"></leaf>
    </container>
    <container name="dateTime">
      <leaf name="enabled" type="boolean" default="true"></leaf>
      <leaf name="format" type="string" default="%Y%m%d.%H%M%S"></leaf>
    </container>
    <container name="fcn">
      <leaf name="enabled" type="boolean" default="true"></leaf>
      <leaf name="format" type="string"
default="`${CLASS}::${FCNNAME}($ARGSS)@$LINE`"></leaf>
    </container>
  </container>
  <container name="dest">
    <container name="console">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
    </container>
    <container name="file">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="name" type="string" default="mtx-internal.log"></leaf>
      <leaf name="location" type="string" default="./mtxlogs">/volatile</leaf>
    </container>
    <leaf name="mbytes-rollover" type="uint32" default="10">50</leaf>
    <leaf name="hours-rollover" type="uint32" default="24">24</leaf>
    <leaf name="startup-rollover" type="boolean" default="false">true</leaf>
    <leaf name="max-rollover-files" type="uint32" default="10">10</leaf>
  </container>
</container>
<list name="logitems" key="id">
  <listitem>
    <leaf name="id" type="string">*</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="false">false</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">SYSTEM</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">LIBUTILS</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">MTX-API</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">Model-*</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">Model-Cisco-NX-OS-device</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">false</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">Model-openconfig-bgp</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">false</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">INST-MTX-API</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">false</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
  </listitem>
  <leaf name="active" type="boolean" default="true">false</leaf>
  </listitem>

```

```
    <listitem>
      <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
    <leaf name="active" type="boolean" default="true">>false</leaf>
  </listitem>
  <listitem>
    <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
    <leaf name="active" type="boolean" default="true">>false</leaf>
  </listitem>
</list>
</container>
</container>
</config>
```




CHAPTER 22

Model-Driven Telemetry

This chapter contains the following topics:

- [About Telemetry, on page 205](#)
- [Configuring Telemetry Using the CLI, on page 213](#)
- [Configuring Telemetry Using the NX-API, on page 228](#)

About Telemetry

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

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

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

Telemetry Components and Process

Telemetry consists of four key elements:

- **Data Collection**—Telemetry data is collected from the Data Management Engine (DME) database in branches of the object model specified using distinguished name (DN) paths. The data can be retrieved periodically (frequency-based) or only when a change occurs in any object on a specified path (event-based). You can use the NX-API to collect frequency-based data.
- **Data Encoding**—The telemetry encoder encapsulates the collected data into the desired format for transporting.
NX-OS encodes telemetry data in the Google Protocol Buffers (GPB) and JSON format.
- **Data Transport**—NX-OS transports telemetry data using HTTP for JSON encoding and the Google remote procedure call (gRPC) protocol for GPB encoding. The gRPC receiver supports message sizes greater than 4 MB. (Telemetry data using HTTPS is also supported if a certificate is configured.)

UDP and secure UDP (DTLS) are supported as telemetry transport protocols. You can add destinations that receive UDP. The encoding for UDP and secure UDP can be GPB or JSON.

Telemetry supports streaming to IPv6 destinations and IPv4 destinations.

Use the following command to configure the UDP transport to stream data using a datagram socket either in JSON or GPB:

```
destination-group num
  ip address xxx.xxx.xxx.xxx port xxxx protocol UDP encoding {JSON | GPB }
```

Example for an IPv4 destination:

```
destination-group 100
  ip address 171.70.55.69 port 50001 protocol UDP encoding GPB
```

Example for an IPv6 destination:

```
destination-group 100
  ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

The UDP telemetry is with the following header:

```
typedef enum tm_encode_ {
    TM_ENCODE_DUMMY,
    TM_ENCODE_GPB,
    TM_ENCODE_JSON,
    TM_ENCODE_XML,
    TM_ENCODE_MAX,
} tm_encode_type_t;

typedef struct tm_pak_hdr_ {
    uint8_t version; /* 1 */
    uint8_t encoding;
    uint16_t msg_size;
    uint8_t secure;
    uint8_t padding;
} __attribute__((packed, aligned(1))) tm_pak_hdr_t;
```

Use the first 6 bytes in the payload to process telemetry data using UDP, using one of the following methods:

- Read the information in the header to determine which decoder to use to decode the data, JSON or GPB, if the receiver is meant to receive different types of data from multiple endpoints.
- Remove the header if you are expecting one decoder (JSON or GPB) but not the other.



Note Depending on the receiving operation system and the network load, using the UDP protocol may result in packet drops.

- **Telemetry Receiver**—A telemetry receiver is a remote management system or application that stores the telemetry data.

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

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

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

High Availability of the Telemetry Process

High availability of the telemetry process has the following behaviors:

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

Licensing Requirements for Telemetry

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

Installing and Upgrading Telemetry

Installing the Application

The telemetry application is packaged as a feature RPM and included with the NX-OS release. The RPM is installed by default as part of the image bootup. After installation, you can start the application using the `feature telemetry` command. The RPM file is in the `/rpms` directory and has the following name:

telemetry-version-build_ID.libn32_n3000.rpm

As in the following example:

```
telemetry-2.0.0.lib32_n3000.rpm
```

Installing Incremental Updates and Fixes

Copy the RPM to the device bootflash and use the following commands from the `bash` prompt:

```
feature bash
run bash sudo su
```

Then copy the RPM to the device bootflash. Use the following commands from the `bash` prompt:

```
yum upgrade telemetry_new_version.rpm
```

When the application restarts, it is upgraded and the change appears.

Downgrading to a Previous Version

To downgrade the telemetry application to a previous version, use the following command from the `bash` prompt:

```
yum downgrade telemetry
```

Verifying the Active Version

To verify the active version, run the following command from the switch `exec` prompt:

```
show install active
```



Note The `show install active` command shows the active installed RPM only after an upgrade has occurred. The default RPM that comes bundled with the NX-OS is not displayed.

Guidelines and Limitations

Telemetry has the following configuration guidelines and limitations:

- Telemetry is supported in Cisco NX-OS releases that support the data management engine (DME) Native Model.
- Telemetry supports DME data collection, NX-API data sources, Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport, and JSON encoding over HTTP.
- The smallest sending interval (cadence) supported is five seconds for a depth of 0. The minimum cadence values for depth values greater than 0 depends on the size of the data being streamed out. Configuring cadences below the minimum value may result in undesirable system behavior.
- Up to five remote management receivers (destinations) are supported. Configuring more than five remote receivers may result in undesirable system behavior.
- If a telemetry receiver goes down, other receivers see data flow interrupted. The failed receiver must be restarted. Then start a new connection with the switch by unconfiguring then reconfiguring the failed receiver's IP address under the destination group.
- Telemetry can consume up to 20% of the CPU resource.
- To configure SSL certificate-based authentication and the encryption of streamed data, you can provide a self-signed SSL certificate with `certificate ssl cert path hostname "CN"` command.

Configuration Commands After Downgrading to an Older Release

After a downgrade to an older release, some configuration commands or command options can fail because the older release may not support them. As a best practice when downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up. By doing so, you avoid possible failure of unsupported commands or command options.

The following example shows this procedure:

- Copy the telemetry configuration to a file:

```
switch# show running-config | section telemetry
```



```

feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config
switch# show file bootflash:telemetry_running_config
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch#

```

- Execute the downgrade operation. When the image comes up and the switch is ready, copy the telemetry configurations back to the switch:

```

switch# copy telemetry_running_config running-config echo-commands
`switch# config terminal`
`switch(config)# feature telemetry`
`switch(config)# telemetry`
`switch(config-telemetry)# destination-group 100`
`switch(config-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(config-tm-dest)# sensor-group 100`
`switch(config-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(config-tm-sensor)# subscription 600`
`switch(config-tm-sub)# dst-grp 100`
`switch(config-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(config-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#

```

gRPC Error Behavior

The switch client disable the connection to the gRPC receiver if the gRPC receiver sends 20 errors. You will need to unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

- The gRPC client sends the wrong certificate for secure connections.
- The gRPC receiver takes too long to handle client messages and incurs a timeout. Avoid timeouts by processing messages using a separate message processing thread.

Telemetry Compression for gRPC Transport

Telemetry compression support is available for gRPC transport. You can use the **use-compression gzip** command to enable compression. (Disable compression with the **no use-compression gzip** command.)

The following example enables compression:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-compression gzip
```

The following example shows that compression is enabled:

```
switch(conf-tm-dest)# show telemetry transport 0 stats

Session Id:                0
Connection Stats
  Connection Count         0
  Last Connected:         Never
  Disconnect Count        0
  Last Disconnected:      Never
Transmission Stats
  Compression:             gzip
  Source Interface:        loopback1(1.1.3.4)
  Transmit Count:          0
  Last TX time:           None
  Min Tx Time:             0 ms
  Max Tx Time:             0 ms
  Avg Tx Time:             0 ms
  Cur Tx Time:             0 ms
```

```
switch2(config-if)# show telemetry transport 0 stats

Session Id: 0
Connection Stats
Connection Count 0
Last Connected: Never
Disconnect Count 0
Last Disconnected: Never
Transmission Stats
Compression: disabled
Source Interface: loopback1(1.1.3.4)
Transmit Count: 0
Last TX time: None
Min Tx Time: 0 ms
Max Tx Time: 0 ms
Avg Tx Time: 0 ms
Cur Tx Time: 0 ms
switch2(config-if)#
```

The following is an example of use-compression as a POST payload:

```
{
  "telemetryDestProfile": {
    "attributes": {
      "adminSt": "enabled"
    },
    "children": [
      {
        "telemetryDestOptCompression": {
          "attributes": {
            "name": "gzip"
          }
        }
      }
    ]
  }
}
```

Support for gRPC Chunking

For streaming to occur successfully, you must enable chunking if gRPC has to send an amount of data greater than 12 MB to the receiver.

gRPC chunking must be done by the gRPC user. Fragmentation occurs on the gRPC client side and reassembly occurs on the gRPC server side. Telemetry is still bound to memory and data can be dropped if the memory size is more than the allowed limit of 12 MB for telemetry. In order to support chunking, use the telemetry .proto file that is available at Cisco's GibLab, which has been updated for gRPC chunking, as described in [Telemetry Components and Process, on page 205](#).

The chunking size is from 64 through 4096 bytes.

Following shows a configuration example through the NX-API CLI:

```
feature telemetry
!
telemetry
  destination-group 1
    ip address 171.68.197.40 port 50051 protocol gRPC encoding GPB
    use-chunking size 4096
  destination-group 2
    ip address 10.155.0.15 port 50001 protocol gRPC encoding GPB
    use-chunking size 64
  sensor-group 1
    path sys/intf depth unbounded
  sensor-group 2
    path sys/intf depth unbounded
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 10000
  subscription 2
    dst-grp 2
    snsr-grp 2 sample-interval 15000
```

Following shows a configuration example through the NX-API REST:

```
{
  "telemetryDestGrpOptChunking": {
    "attributes": {
      "chunkSize": "2048",
      "dn": "sys/tm/dest-1/chunking"
    }
  }
}
```

The following error message appears on systems that do not support gRPC chunking:

```
switch-1(conf-tm-dest)# use-chunking size 200
ERROR: Operation failed: [chunking support not available]
```

NX-API Sensor Path Limitations

NX-API can collect and stream switch information not yet in the DME using **show** commands. However, using the NX-API instead of streaming data from the DME has inherent scale limitations as outlined:

- The switch backend dynamically processes NX-API calls such as **show** commands,
- NX-API spawns several processes that can consume up to a maximum of 20% of the CPU.
- NX-API data translates from the CLI to XML to JSON.

The following is a suggested user flow to help limit excessive NX-API sensor path bandwidth consumption:

1. Check whether the **show** command has NX-API support. You can confirm whether NX-API supports the command from the VSH with the pipe option: `show <command> | json` OR `show <command> | json pretty`.



Note Avoid commands that take the switch more than 30 seconds to return JSON output.

2. Refine the **show** command to include any filters or options.
 - Avoid enumerating the same command for individual outputs; for example, **show vlan id 100** , **show vlan id 101** , and so on. Instead, use the CLI range options; for example, **show vlan id 100-110,204** , whenever possible to improve performance.

If you need only the summary or counter, avoid dumping a whole show command output. By doing so, you limit the bandwidth and data storage that is required for data collection.
3. Configure telemetry with sensor groups that use NX-API as their data sources. Add the **show** commands as sensor paths
4. Configure telemetry with a cadence of five times the processing time of the respective **show** command to limit CPI usage.
5. Receive and process the streamed NX-API output as part of the existing DME collection.

Telemetry VRF Support

Telemetry VRF support allows you to specify a transport VRF, which means that the telemetry data stream can egress through front-panel ports and avoid possible competition between SSH/NGINX control sessions.

You can use the **use-vrf vrf-name** command to specify the transport VRF.

The following example specifies the transport VRF:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-vrf test_vrf
```

The following is an example of use-vrf as a POST payload:

```
{
  "telemetryDestProfile": {
    "attributes": {
      "adminSt": "enabled"
    },
    "children": [
      {
        "telemetryDestOptVrf": {
          "attributes": {
            "name": "default"
          }
        }
      }
    ]
  }
}
```

```
    }
}
```

Support for Streaming of YANG Models

The YANG ("Yet Another Next Generation") data modeling language is supported as part of telemetry. Both device YANG and open config YANG model data streaming are supported.

Configuring Telemetry Using the CLI

Configuring Telemetry Using the NX-OS CLI

The following steps enable streaming telemetry and configuring the source and destination of the data stream. These steps also include optional steps to enable and configure SSL/TLS certificates and GPB encoding.

Procedure

	Command or Action	Purpose
Step 1	<p>(Optional) <code>openssl argument</code></p> <p>Example:</p> <p>Generate an SSL/TLS certificate using a specific argument, such as the following:</p> <ul style="list-style-type: none"> To generate a private RSA key: <code>openssl genrsa -cipher -out filename.key cipher-bit-length</code> <p>For example:</p> <pre>switch# openssl genrsa -des3 -out server.key 2048</pre> <ul style="list-style-type: none"> To write the RSA key: <code>openssl rsa -in filename.key -out filename.key</code> <p>For example:</p> <pre>switch# openssl rsa -in server.key -out server.key</pre> <ul style="list-style-type: none"> To create a certificate that contains the public or private key: <code>openssl req -encoding-standard -new -new filename.key -out filename.csr -subj '/CN=localhost'</code> <p>For example:</p> <pre>switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'</pre>	<p>Create an SSL or TLS certificate on the server that will receive the data, where <code>private.key</code> file is the private key and the <code>public.crt</code> is the public key.</p>

	Command or Action	Purpose
	<ul style="list-style-type: none"> To create a public key: <code>openssl x509 -req -encoding-standard -days timeframe -in filename.csr -signkey filename.key -out filename.csr</code> <p>For example:</p> <pre>switch# openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt</pre>	
Step 2	configure terminal Example: <pre>switch# configure terminal switch(config)#</pre>	Enter the global configuration mode.
Step 3	feature telemetry	Enable the streaming telemetry feature.
Step 4	feature nxapi	Enable NX-API.
Step 5	nxapi use-vrf management	Enable the VRF management to be used for NX-API communication.
Step 6	telemetry Example: <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	Enter configuration mode for streaming telemetry.
Step 7	(Optional) certificate <i>certificate_path</i> <i>host_URL</i> Example: <pre>switch(config-telemetry)# certificate /bootflash/server.key localhost</pre>	Use an existing SSL/TLS certificate.
Step 8	(Optional) Specify a transport VRF or enable telemetry compression for gRPC transport. Example: <pre>switch(config-telemetry)# destination-profile switch(conf-tm-dest-profile)# use-vrf default switch(conf-tm-dest-profile)# use-compression gzip switch(conf-tm-dest-profile)# use-retry size 10 switch(conf-tm-dest-profile)# source-interface loopback1</pre>	<ul style="list-style-type: none"> Enter the destination-profile command to specify the default destination profile. Enter any of the following commands: <ul style="list-style-type: none"> use-vrf <i>vrf</i> to specify the destination VRF. use-compression gzip to specify the destination compression method. use-retry size <i>size</i> to specify the send retry details, with a retry buffer size from 10 through 1500 megabytes. source-interface <i>interface-name</i> to stream data from the configured

	Command or Action	Purpose
		<p>interface to a destination with the source IP address.</p> <p>Note After configuring the use-vrf command, you must configure a new destination IP address within the new VRF. However, you may re-use the same destination IP address by unconfiguring and reconfiguring the destination. This ensures that the telemetry data streams to the same destination IP address in the new VRF.</p>
Step 9	<p>sensor-group <i>sgrp_id</i></p> <p>Example:</p> <pre>switch(config-telemetry)# sensor-group 100 switch(conf-tm-sensor)#</pre>	<p>Create a sensor group with ID <i>sgrp_id</i> and enter sensor group configuration mode.</p> <p>Currently only numeric ID values are supported. The sensor group defines nodes that will be monitored for telemetry reporting.</p>
Step 10	<p>(Optional) data-source <i>data-source-type</i></p> <p>Example:</p> <pre>switch(config-telemetry)# data-source NX-API</pre>	<p>Select a data source. Select from either YANG, DME or NX-API as the data source.</p> <p>Note DME is the default data source.</p>
Step 11	<p>path <i>sensor_path</i> depth 0 [filter-condition <i>filter</i>]</p> <p>Example:</p> <ul style="list-style-type: none"> The following command is applicable for DME, not for NX-API or YANG: <pre>switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(12BD.operSt, "down")</pre> <p>Use the following syntax for state-based filtering to trigger only when operSt changes from up to down, with no notifications of when the MO changes.</p> <pre>switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(12BD.operSt),eq(12BD.operSt,"down"))</pre> The following command is applicable for NX-API, not for DME or YANG: <pre>switch(conf-tm-sensor)# path "show interface" depth 0</pre> 	<p>Add a sensor path to the sensor group.</p> <ul style="list-style-type: none"> The depth setting specifies the retrieval level for the sensor path. Depth settings of 0 - 32, unbounded are supported. <p>Note depth 0 is the default depth.</p> <p>NX-API-based sensor paths can only use depth 0.</p> <p>If a path is subscribed for the event collection, the depth only supports 0 and unbounded. Other values are treated as 0.</p> <ul style="list-style-type: none"> The optional filter-condition parameter can be specified to create a specific filter for event-based subscriptions. <p>For state-based filtering, the filter returns both when a state has changed and when an event has occurred during the specified state. That is, a filter condition for the DN sys/bd/bd-[vlan] of eq(12Bd.operSt,</p>

	Command or Action	Purpose
	<ul style="list-style-type: none"> The following command is applicable for device YANG: <pre>switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items</pre> The following command is applicable for OpenConfig YANG: <pre>switch(conf-tm-sensor)# path openconfig-bgp:bgp</pre> 	<p>"down") triggers when the operSt changes, and when the DN's property changes while the operSt remains down. One example is when a no shutdown command is issued while the VLAN is operationally down.</p> <p>Note query-condition parameter—For DME, based on the DN, the query-condition parameter can be specified to fetch MOTL and ephemeral data with the following syntax: query-condition "rsp-foreign-subtree=applied-config"; query-condition "rsp-foreign-subtree=ephemeral".</p> <ul style="list-style-type: none"> For the YANG model, the sensor path format is as follows: <i>module_name</i>:<i>YANG_path</i>, where <i>module_name</i> is the name of the YANG model file. For example: <ul style="list-style-type: none"> For device YANG: <pre>Cisco-NX-OS-device:System/bgp-items/inst-items</pre> For OpenConfig YANG: <pre>openconfig-bgp:bgp</pre> <p>Note The depth, filter-condition, and query-condition parameters are not supported for YANG currently.</p> <p>For the openconfig YANG models, go to https://github.com/YangModels/yang/tree/master/vendor/cisco/nx and navigate to the appropriate folder for the latest release.</p> <p>All the openconfig YANG models have a specific RPM, so you must install the associated RPM before you can use telemetry. See Adding Patch RPMs from Bash, on page 19 for more information on installing patch RPMs.</p>
Step 12	<pre>destination-group <i>dgrp_id</i></pre> <p>Example:</p>	Create a destination group and enter destination group configuration mode.

	Command or Action	Purpose
	<pre>switch(conf-tm-sensor) # destination-group 100 switch(conf-tm-dest) #</pre>	Currently <i>dgrp_id</i> only supports numeric ID values.
Step 13	<p>(Optional) ip address <i>ip_address</i> port <i>port</i> protocol <i>procedural-protocol</i> encoding <i>encoding-protocol</i></p> <p>Example:</p> <pre>switch(conf-tm-sensor) # ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB switch(conf-tm-sensor) # ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON switch(conf-tm-sensor) # ip address 171.70.55.69 port 50009 protocol UDP encoding JSON</pre>	<p>Specify an IPv4 IP address and port to receive encoded telemetry data.</p> <p>Note gRPC is the default transport protocol. GPB is the default encoding.</p>
Step 14	<p>(Optional) ipv6 address <i>ipv6_address</i> port <i>port</i> protocol <i>procedural-protocol</i> encoding <i>encoding-protocol</i></p> <p>Example:</p> <pre>switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON</pre>	<p>Specify an IPv6 IP address and port to receive encoded telemetry data.</p> <p>Note gRPC is the default transport protocol. GPB is the default encoding.</p>
Step 15	<p>ip_version address <i>ip_address</i> port <i>portnum</i></p> <p>Example:</p> <ul style="list-style-type: none"> For IPv4: <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50003</pre> For IPv6: <pre>switch(conf-tm-dest) # ipv6 address 10:10::1 port 8000</pre> 	<p>Create a destination profile for the outgoing data, where <i>ip_version</i> is either ip (for IPv4) or ipv6 (for IPv6).</p> <p>When the destination group is linked to a subscription, telemetry data is sent to the IP address and port that the profile specifies.</p>
Step 16	<p>(Optional) use-chunking size <i>chunking_size</i></p> <p>Example:</p> <pre>switch(conf-tm-dest) # use-chunking size 64</pre>	<p>Enable gRPC chunking and set the chunking size, from 64 through 4096 bytes. See the section "Support for gRPC Chunking" for more information.</p>

	Command or Action	Purpose
Step 17	subscription <i>sub_id</i> Example: <pre>switch(conf-tm-dest)# subscription 100 switch(conf-tm-sub)#</pre>	<p>Create a subscription node with ID and enter the subscription configuration mode.</p> <p>Currently <i>sub_id</i> only supports numeric ID values.</p> <p>Note When subscribing to a DN, check whether the DN is supported by DME using REST to ensure that events can stream.</p>
Step 18	snsr-grp <i>sgrp_id</i> sample-interval <i>interval</i> Example: <pre>switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000</pre>	<p>Link the sensor group with ID <i>sgrp_id</i> to this subscription and set the data sampling interval in milliseconds.</p> <p>An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds.</p>
Step 19	dst-grp <i>dgrp_id</i> Example: <pre>switch(conf-tm-sub)# dst-grp 100</pre>	<p>Link the destination group with ID <i>dgrp_id</i> to this subscription.</p>

Configuration Examples for Telemetry Using the CLI

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

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data for `sys/intf` every 5 seconds to destination IP 1.2.3.4 port 50003. The subscription encrypts the stream using GPB encoding that is verified using the `test.pem`.

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(conf-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
```

```
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data for `sys/cdp` every 15 seconds to destination IP 1.2.3.4 port 50004.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

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

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor)# data-source NX-API
switch(conf-tm-sensor)# path "show system resources" depth 0
switch(conf-tm-sensor)# path "show version" depth 0
switch(conf-tm-sensor)# path "show environment power" depth 0
switch(conf-tm-sensor)# path "show environment fan" depth 0
switch(conf-tm-sensor)# path "show environment temperature" depth 0
switch(conf-tm-sensor)# path "show process cpu" depth 0
switch(conf-tm-sensor)# path "show nve peers" depth 0
switch(conf-tm-sensor)# path "show nve vni" depth 0
switch(conf-tm-sensor)# path "show nve vni 4002 counters" depth 0
switch(conf-tm-sensor)# path "show int nve 1 counters" depth 0
switch(conf-tm-sensor)# path "show policy-map vlan" depth 0
switch(conf-tm-sensor)# path "show ip access-list test" depth 0
switch(conf-tm-sensor)# path "show system internal access-list resource utilization" depth
0
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

This example creates an event-based subscription for `sys/fm`. Data is streamed to the destination only if there is a change under the `sys/fm` MO.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
```

During operation, you can change a sensor group from frequency-based to event-based, and change event-based to frequency-based by changing the `sample-interval`. This example changes the sensor-group from the previous example to frequency-based. After the following commands, the telemetry application will begin streaming the `sys/fm` data to the destination every 7 seconds.

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

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

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

A sensor group can contain multiple paths. A destination group can contain multiple destination profiles. You can link a subscription to multiple sensor groups and destination groups, as shown in the following example.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# path sys/epId-1 depth 0
switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# path sys/bgp depth 0

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

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

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

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

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

```
switch(conf-tm-sub)# dst-grp 300
```

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

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

!Command: show running-config telemetry
!Running configuration last done at: Wed Apr 12 21:29:11 2000
!Time: Wed Apr 12 21:29:23 2000

version 9.2(1) Bios:version 05.35
feature telemetry

telemetry

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

You can specify transport VRF and telemetry data compression for gRPC using the **use-vrf** and **use-compression gzip** commands, as shown in this example.

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-vrf default
switch(conf-tm-dest-profile)# use-compression gzip
switch(conf-tm-dest-profile)# sensor-group 1
switch(conf-tm-sensor)# path sys/bgp depth unbounded
switch(conf-tm-sensor)# destination-group 1
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-sub)# snsr-grp 1 sample-interval 10000
```

Displaying Telemetry Configuration and Statistics

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

show telemetry control database

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

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

```

subscriptions      Show subscriptions
|                  Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1

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

Sensor Group Database size = 1

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

Sensor Path Database size = 1

-----
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
-----
No                      1              0           Full            sys/fm

Destination group Database size = 2

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

Destination Database size = 2

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

```

show telemetry control stats

This command displays the statistics about the internal databases about configuration of telemetry.

```

switch# show telemetry control stats
show telemetry control stats entered

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

```

```

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

```

```

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

```

show telemetry data collector brief

This command displays the brief statistics about the data collection.

```
switch# show telemetry data collector brief
```

```

-----
Collector Type      Successful Collections   Failed Collections
-----
DME                  143                       0

```

show telemetry data collector details

This command displays detailed statistics about the data collection which includes breakdown of all sensor paths.

```
switch# show telemetry data collector details
```

```
-----
Succ Collections      Failed Collections    Sensor Path
-----
150                   0                     sys/fm
-----
```

show telemetry event collector errors

This command displays the errors statistic about the event collection.

```
switch# show telemetry event collector errors
```

```
-----
Error Description                                          Error Count
-----
APIC-Cookie Generation Failures                          - 0
Authentication Failures                                  - 0
Authentication Refresh Failures                          - 0
Authentication Refresh Timer Start Failures              - 0
Connection Timer Start Failures                          - 0
Connection Attempts                                      - 3
Dme Event Subscription Init Failures                     - 0
Event Data Enqueue Failures                              - 0
Event Subscription Failures                              - 0
Event Subscription Refresh Failures                      - 0
Pending Subscription List Create Failures                 - 0
Subscription Hash Table Create Failures                   - 0
Subscription Hash Table Destroy Failures                  - 0
Subscription Hash Table Insert Failures                  - 0
Subscription Hash Table Remove Failures                  - 0
Subscription Refresh Timer Start Failures                 - 0
Websocket Connect Failures                               - 0
-----
```

show telemetry event collector stats

This command displays the statistics about the event collection which includes breakdown of all sensor paths.

```
switch# show telemetry event collector stats
```

```
-----
Collection Count  Latest Collection Time    Sensor Path
-----
```

show telemetry control pipeline stats

This command displays the statistics for the telemetry pipeline.

```
switch# show telemetry pipeline stats
Main Statistics:
  Timers:
```



```

Errors:
  Start Fail          =    0

Data Collector:
  Errors:
    Node Create Fail  =    0

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

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

      Errors:
        Enqueue Error  =    0     Dequeue Error   =    0

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

      Errors:
        Enqueue Error  =    0     Dequeue Error   =    0

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

      Errors:
        Enqueue Error  =    0     Dequeue Error   =    0

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

      Errors:
        Enqueue Error  =    0     Dequeue Error   =    0

```

show telemetry transport

This command displays all configured transport sessions.

```
switch# show telemetry transport
```

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

```

Retry buffer Size:          10485760
Event Retry Messages (Bytes): 0
Timer Retry Messages (Bytes): 0
Total Retries sent:        0

```

```
Total Retries Dropped:          0
```

show telemetry transport <session-id>

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

```
switch# show telemetry transport 0

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

```
switch# show telemetry transport 1

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

The following example shows output from an IPv6 entry.

```
switch# show telemetry transport 0
Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

show telemetry transport <session-id> stats

This command displays details of a specific transport session.

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:     Mon May 01 11:29:46.912 PST
Last Disconnected:  Never
```

```
Tx Error Count:      0
Last Tx Error:      None
```

show telemetry transport <session-id> errors

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

```
switch# show telemetry transport 0 errors

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

Displaying Telemetry Log and Trace Information

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

show tech-support telemetry

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

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

show system internal telemetry trace

The **show system internal telemetry trace** [**tm-events** | **tm-errors** | **tm-logs** | **all**] command displays system internal telemetry trace information.

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy dest
  profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
  destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
  1 does not have any sensor groups

3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
  initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
  successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
```

```

grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#
switch# show system internal telemetry trace tm-events
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
  initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
  successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
  cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
  cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#

```

Configuring Telemetry Using the NX-API

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in [Telemetry Model in the DME, on page 240](#). Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
 - **fmNxapi** — Contains the NX-API state.
 - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
 - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
 - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
 - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
 - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
 - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.
 - **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.

- **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
 - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
 - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.
- **telemetryCertificate** — Associates the telemetry subscription with a certificate and hostname.

To configure telemetry using the NX-API, you construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.



Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be configured to run the NX-API from the CLI:

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

```
nxapi use-vrf vrf_name
nxapi http port port_number
```

Procedure

	Command or Action	Purpose
Step 1	Enable the telemetry feature. Example: <pre>{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } } }] }</pre>	The root element is fmTelemetry and the base path for this element is <code>sys/fm</code> . Configure the adminSt attribute as <code>enabled</code> .
Step 2	Create the root level of the JSON payload to describe the telemetry configuration. Example: <pre>{ "telemetryEntity": { "attributes": { "dn": "sys/tm" } } }</pre>	The root element is telemetryEntity and the base path for this element is <code>sys/tm</code> . Configure the dn attribute as <code>sys/tm</code> .

	Command or Action	Purpose
	<pre> }, } } } </pre>	
Step 3	<p>Create a sensor group to contain the defined sensor paths.</p> <p>Example:</p> <pre> "telemetrySensorGroup": { "attributes": { "id": "10", "rn": "sensor-10" "dataSrc": "NX-API" }, "children": [{ }] } </pre>	<p>A telemetry sensor group is defined in an object of class telemetrySensorGroup. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> • id — An identifier for the sensor group. Currently only numeric ID values are supported. • rn — The relative name of the sensor group object in the format: sensor-<i>id</i>. • dataSrc — Selects the data source from DEFAULT, DME, YANG, or NX-API. <p>Children of the sensor group object will include sensor paths and one or more relation objects (telemetryRtSensorGroupRel) to associate the sensor group with a telemetry subscription.</p>
Step 4	<p>(Optional) Add an SSL/TLS certificate and a host.</p> <p>Example:</p> <pre> { "telemetryCertificate": { "attributes": { "filename": "root.pem" "hostname": "c.com" } } } </pre>	<p>The telemetryCertificate defines the location of the SSL/TLS certificate with the telemetry subscription/destination.</p>
Step 5	<p>Define a telemetry destination group.</p> <p>Example:</p> <pre> { "telemetryDestGroup": { "attributes": { "id": "20" } } } </pre>	<p>A telemetry destination group is defined in telemetryEntity. Configure the id attribute.</p>
Step 6	<p>Define a telemetry destination profile.</p> <p>Example:</p> <pre> { "telemetryDestProfile": { </pre>	<p>A telemetry destination profile is defined in telemetryDestProfile.</p> <ul style="list-style-type: none"> • Configure the adminSt attribute as enabled.

	Command or Action	Purpose
	<pre> "attributes": { "adminSt": "enabled" }, "children": [{ "telemetryDestOptSourceInterface": { "attributes": { "name": "lo0" } } }] } } } </pre>	<ul style="list-style-type: none"> • Under telemetryDestOptSourceInterface, configure the name attribute with an interface name to stream data from the configured interface to a destination with the source IP address.
Step 7	<p>Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.</p> <p>Example:</p> <pre> { "telemetryDest": { "attributes": { "addr": "1.2.3.4", "enc": "GPB", "port": "50001", "proto": "gRPC", "rn": "addr-[1.2.3.4]-port-50001" } } } </pre>	<p>A telemetry destination is defined in an object of class telemetryDest. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> • addr — The IP address of the destination. • port — The port number of the destination. • rn — The relative name of the destination object in the format: path-[path]. • enc — The encoding type of the telemetry data to be sent. NX-OS supports: <ul style="list-style-type: none"> • Google protocol buffers (GPB) for gRPC. • JSON for C. • GPB or JSON for UDP and secure UDP (DTLS). • proto — The transport protocol type of the telemetry data to be sent. NX-OS supports: <ul style="list-style-type: none"> • gRPC • HTTP • VUDP and secure UDP (DTLS)
Step 8	<p>Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.</p> <p>Example:</p> <pre> { "telemetryDestGrpOptChunking": { </pre>	<p>See "Support for gRPC Chunking" in Guidelines and Limitations, on page 208 for more information.</p>

	Command or Action	Purpose
	<pre> "attributes": { "chunkSize": "2048", "dn": "sys/tm/dest-1/chunking" } } } </pre>	
Step 9	<p>Create a telemetry subscription to configure the telemetry behavior.</p> <p>Example:</p> <pre> "telemetrySubscription": { "attributes": { "id": "30", "rn": "subs-30" }, "children": [{ }] } </pre>	<p>A telemetry subscription is defined in an object of class telemetrySubscription. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> • id — An identifier for the subscription. Currently only numeric ID values are supported. • rn — The relative name of the subscription object in the format: subs-<i>id</i>. <p>Children of the subscription object will include relation objects for sensor groups (telemetryRsSensorGroupRel) and destination groups (telemetryRsDestGroupRel).</p>
Step 10	<p>Add the sensor group object as a child object to the telemetrySubscription element under the root element (telemetryEntity).</p> <p>Example:</p> <pre> { "telemetrySubscription": { "attributes": { "id": "30" } }, "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } }] } } </pre>	
Step 11	<p>Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior.</p> <p>Example:</p>	<p>The relation object is of class telemetryRsSensorGroupRel and is a child object of telemetrySubscription. Configure the following attributes of the relation object:</p>

	Command or Action	Purpose
	<pre> "telemetryRsSensorGroupRel": { "attributes": { "rType": "mo", "rn": "rssensorGroupRel-[sys/tm/sensor-10]", "sampleIntvl": "5000", "tCl": "telemetrySensorGroup", "tDn": "sys/tm/sensor-10", "tType": "mo" } } </pre>	<ul style="list-style-type: none"> • rn — The relative name of the relation object in the format: rssensorGroupRel-[sys/tm/sensor-group-id]. • sampleIntvl — The data sampling period in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds. • tCl — The class of the target (sensor group) object, which is telemetrySensorGroup. • tDn — The distinguished name of the target (sensor group) object, which is sys/tm/sensor-group-id. • rType — The relation type, which is mo for managed object. • tType — The target type, which is mo for managed object.
<p>Step 12</p>	<p>Define one or more sensor paths or nodes to be monitored for telemetry.</p> <p>Example:</p> <p>Single sensor path</p> <pre> { "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } } </pre> <p>Example:</p>	<p>A sensor path is defined in an object of class telemetrySensorPath. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> • path — The path to be monitored. • rn — The relative name of the path object in the format: path-[path] • depth — The retrieval level for the sensor path. A depth setting of 0 retrieves only the root MO properties. • filterCondition — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_

Command or Action	Purpose
<p>Single sensor path for NX-API</p> <pre data-bbox="500 352 971 730"> { "telemetrySensorPath": { "attributes": { "path": "show interface", "path": "show bgp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } } </pre> <p>Example: Multiple sensor paths</p> <pre data-bbox="500 890 971 1549"> { "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }, { "telemetrySensorPath": { "attributes": { "excludeFilter": "", "filterCondition": "", "path": "sys/fm/dhcp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } } </pre> <p>Example: Single sensor path filtering for BGP disable events:</p> <pre data-bbox="500 1738 971 1852"> { "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", </pre>	<p>REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635</p>

	Command or Action	Purpose
	<pre> "excludeFilter": "", "filterCondition": "eq(fmBgp.operSt.\"disabled\")", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } } </pre>	
Step 13	Add sensor paths as child objects to the sensor group object (telemetrySensorGroup).	
Step 14	Add destinations as child objects to the destination group object (telemetryDestGroup).	
Step 15	Add the destination group object as a child object to the root element (telemetryEntity).	
Step 16	<p>Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.</p> <p>Example:</p> <pre> "telemetryRtSensorGroupRel": { "attributes": { "rn": "rtsensorGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } } </pre>	<p>The relation object is of class telemetryRtSensorGroupRel and is a child object of telemetrySensorGroup. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> • rn — The relative name of the relation object in the format: rtsensorGroupRel-[sys/tm/subscription-id]. • tCl — The target class of the subscription object, which is telemetrySubscription. • tDn — The target distinguished name of the subscription object, which is sys/tm/subscription-id.
Step 17	<p>Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.</p> <p>Example:</p> <pre> "telemetryRtDestGroupRel": { "attributes": { "rn": "rtdestGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } } </pre>	<p>The relation object is of class telemetryRtDestGroupRel and is a child object of telemetryDestGroup. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> • rn — The relative name of the relation object in the format: rtdestGroupRel-[sys/tm/subscription-id]. • tCl — The target class of the subscription object, which is telemetrySubscription. • tDn — The target distinguished name of the subscription object, which is sys/tm/subscription-id.

	Command or Action	Purpose
Step 18	<p>Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.</p> <p>Example:</p> <pre> "telemetryRsDestGroupRel": { "attributes": { "rType": "mo", "rn": "rsdestGroupRel-[sys/tm/dest-20]", "tCl": "telemetryDestGroup", "tDn": "sys/tm/dest-20", "tType": "mo" } } </pre>	<p>The relation object is of class telemetryRsDestGroupRel and is a child object of telemetrySubscription. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> • rn — The relative name of the relation object in the format: rsdestGroupRel-[sys/tm/destination-group-id]. • tCl — The class of the target (destination group) object, which is telemetryDestGroup. • tDn — The distinguished name of the target (destination group) object, which is sys/tm/destination-group-id. • rType — The relation type, which is mo for managed object. • tType — The target type, which is mo for managed object.
Step 19	<p>Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.</p>	<p>The base path for the telemetry entity is <code>sys/tm</code> and the NX-API endpoint is:</p> <pre>{{URL}}/api/node/mo/sys/tm.json</pre>

Example

The following is an example of all the previous steps collected into one POST payload (note that some attributes may not match):

```

{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
      }
    ]
  }
},
{
  "telemetrySensorPath": {
    "attributes": {
      "excludeFilter": "",
      "filterCondition": "",
      "path": "sys/fm/bgp",
      "secondaryGroup": "0",
      "secondaryPath": "",
      "depth": "0"
    }
  }
}

```

```

    "telemetryDestGroup": {
      "attributes": {
        "id": "20"
      }
      "children": [{
        "telemetryDest": {
          "attributes": {
            "addr": "10.30.217.80",
            "port": "50051",
            "enc": "GPB",
            "proto": "gRPC"
          }
        }
      ]
    }
  },
  {
    "telemetrySubscription": {
      "attributes": {
        "id": "30"
      }
      "children": [{
        "telemetryRsSensorGroupRel": {
          "attributes": {
            "sampleIntvl": "5000",
            "tDn": "sys/tm/sensor-10"
          }
        }
      ],
      {
        "telemetryRsDestGroupRel": {
          "attributes": {
            "tDn": "sys/tm/dest-20"
          }
        }
      ]
    }
  }
]
}
}

```

Configuration Example for Telemetry Using the NX-API

Streaming Paths to a Destination

This example creates a subscription that streams paths `sys/cdp` and `sys/ipv4` to a destination `1.2.3.4 port 50001` every five seconds.

POST <https://192.168.20.123/api/node/mo/sys/tm.json>

Payload:

```

{
  "telemetryEntity": {
    "attributes": {
      "dn": "sys/tm"
    },
    "children": [{

```

```

"telemetrySensorGroup": {
  "attributes": {
    "id": "10",
    "rn": "sensor-10"
  },
  "children": [{
    "telemetryRtSensorGroupRel": {
      "attributes": {
        "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
        "tCl": "telemetrySubscription",
        "tDn": "sys/tm/subs-30"
      }
    }
  ]
}, {
  "telemetrySensorPath": {
    "attributes": {
      "path": "sys/cdp",
      "rn": "path-[sys/cdp]",
      "excludeFilter": "",
      "filterCondition": "",
      "secondaryGroup": "0",
      "secondaryPath": "",
      "depth": "0"
    }
  }
}, {
  "telemetrySensorPath": {
    "attributes": {
      "path": "sys/ipv4",
      "rn": "path-[sys/ipv4]",
      "excludeFilter": "",
      "filterCondition": "",
      "secondaryGroup": "0",
      "secondaryPath": "",
      "depth": "0"
    }
  }
}
]]
}, {
  "telemetryDestGroup": {
    "attributes": {
      "id": "20",
      "rn": "dest-20"
    },
    "children": [{
      "telemetryRtDestGroupRel": {
        "attributes": {
          "rn": "rtdestGroupRel-[sys/tm/subs-30]",
          "tCl": "telemetrySubscription",
          "tDn": "sys/tm/subs-30"
        }
      }
    ]
  }, {
    "telemetryDest": {
      "attributes": {
        "addr": "1.2.3.4",
        "enc": "GPB",
        "port": "50001",
        "proto": "gRPC",
        "rn": "addr-[1.2.3.4]-port-50001"
      }
    }
  }
}
]]
}

```

```

    }, {
      "telemetrySubscription": {
        "attributes": {
          "id": "30",
          "rn": "subs-30"
        },
        "children": [{
          "telemetryRsDestGroupRel": {
            "attributes": {
              "rType": "mo",
              "rn": "rsdestGroupRel-[sys/tm/dest-20]",
              "tCl": "telemetryDestGroup",
              "tDn": "sys/tm/dest-20",
              "tType": "mo"
            }
          }
        }, {
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "rType": "mo",
              "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
              "sampleIntvl": "5000",
              "tCl": "telemetrySensorGroup",
              "tDn": "sys/tm/sensor-10",
              "tType": "mo"
            }
          }
        }
      ]
    }
  ]
}

```

Filter Conditions on BGP Notifications

The following example payload enables notifications that trigger when the BFP feature is disabled as per the `filterCondition` attribute in the `telemetrySensorPath` MO. The data is streamed to 10.30.217.80 port 50055.

POST <https://192.168.20.123/api/node/mo/sys/tm.json>

Payload:

```

{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
      },
      "children": [{
        "telemetrySensorPath": {
          "attributes": {
            "excludeFilter": "",
            "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
          }
        }
      ]
    }
  ]
}

```

```

    }
  },
  {
    "telemetryDestGroup": {
      "attributes": {
        "id": "20"
      }
      "children": [{
        "telemetryDest": {
          "attributes": {
            "addr": "10.30.217.80",
            "port": "50055",
            "enc": "GPB",
            "proto": "gRPC"
          }
        }
      ]
    }
  },
  {
    "telemetrySubscription": {
      "attributes": {
        "id": "30"
      }
      "children": [{
        "telemetryRsSensorGroupRel": {
          "attributes": {
            "sampleIntvl": "0",
            "tDn": "sys/tm/sensor-10"
          }
        }
      ],
      {
        "telemetryRsDestGroupRel": {
          "attributes": {
            "tDn": "sys/tm/dest-20"
          }
        }
      ]
    }
  }
]
}
}

```

Using Postman Collection for Telemetry Configuration

An [example Postman collection](#) is an easy way to start configuring the telemetry feature, and can run all telemetry CLI equivalents in a single payload. Modify the file in the preceding link using your preferred text editor to update the payload to your needs, then open the collection in Postman and run the collection.

Telemetry Model in the DME

The telemetry application is modeled in the DME with the following structure:

```

model
|----package [name:telemetry]
|   @name:telemetry
|----objects

```



```

|----mo [name:Entity]
|   @name:Entity
|   @label:Telemetry System
|--property
|   @name:adminSt
|   @type:AdminState
|
|----mo [name:SensorGroup]
|   |   @name:SensorGroup
|   |   @label:Sensor Group
|   |--property
|   |   @name:id [key]
|   |   @type:string:Basic
|   |   @name:dataSrc
|   |   @type:DataSource
|   |
|   |----mo [name:SensorPath]
|   |   |   @name:SensorPath
|   |   |   @label:Sensor Path
|   |   |--property
|   |   |   @name:path [key]
|   |   |   @type:string:Basic
|   |   |   @name:filterCondition
|   |   |   @type:string:Basic
|   |   |   @name:excludeFilter
|   |   |   @type:string:Basic
|   |   |   @name:depth
|   |   |   @type:RetrieveDepth
|   |
|   |----mo [name:DestGroup]
|   |   |   @name:DestGroup
|   |   |   @label:Destination Group
|   |   |--property
|   |   |   @name:id
|   |   |   @type:string:Basic
|   |   |
|   |   |----mo [name:Dest]
|   |   |   |   @name:Dest
|   |   |   |   @label:Destination
|   |   |   |--property
|   |   |   |   @name:addr [key]
|   |   |   |   @type:address:Ip
|   |   |   |   @name:port [key]
|   |   |   |   @type:scalar:Uint16
|   |   |   |   @name:proto
|   |   |   |   @type:Protocol
|   |   |   |   @name:enc
|   |   |   |   @type:Encoding
|   |   |
|   |   |----mo [name:Subscription]
|   |   |   |   @name:Subscription
|   |   |   |   @label:Subscription
|   |   |   |--property
|   |   |   |   @name:id
|   |   |   |   @type:scalar:Uint64
|   |   |   |----reldef
|   |   |   |   |   @name:SensorGroupRel
|   |   |   |   |   @to:SensorGroup
|   |   |   |   |   @cardinality:ntom
|   |   |   |   |   @label:Link to sensorGroup entry
|   |   |   |   |--property
|   |   |   |   |   @name:sampleIntvl
|   |   |   |   |   @type:scalar:Uint64
|   |   |   |
|   |   |
|   |
|

```

```
|----reldef
|   @name:DestGroupRel
|   @to:DestGroup
|   @cardinality:ntom
|   @label:Link to destGroup entry
```



PART **V**

XML Mangement Interface

- [XML Management Interface, on page 245](#)



CHAPTER 23

XML Management Interface

This chapter contains the following topics:

- [About the XML Management Interface, on page 245](#)
- [Information About Example XML Instances, on page 260](#)
- [Additional References, on page 267](#)

About the XML Management Interface

Information About the XML Management Interface

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

NETCONF is implemented with an XML Schema (XSD) that allows you to enclose device configuration elements within a remote procedure call (RPC) message. From within an RPC message, select one of the NETCONF operations that matches the type of command that you want the device to execute. You can configure the entire set of CLI commands on the device with NETCONF. For information about using NETCONF, see [Creating NETCONF XML Instances, on page 250](#) and [RFC 4741](#).

For more information about using NETCONF over SSH, see [RFC 4742](#).

This section includes the following topics:

NETCONF Layers

The following table lists the NETCONF layers:

Table 15: NETCONF Layers

Layer	Example
Transport protocol	SSHv2
RPC	RPC, RPC-reply
Operations	get-config, edit-config

Layer	Example
Content	show or configuration command

The following is a description of the four NETCONF layers:

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

SSH xmlagent

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



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

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

The XML schemas that define the XML configuration instances that you can use are described in [Creating NETCONF XML Instances, on page 250](#).

Licensing Requirements for the XML Management Interface

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

Prerequisites to Using the XML Management Interface

Using the XML management interface has the following prerequisites:

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

Using the XML Management Interface

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



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

Configuring the SSH and the XML Server Options Through the CLI

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

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



Note The XML server timeout applies only to active sessions.

For more information about configuring SSH, see the Cisco NX-OS security configuration guide for your platform.

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

Procedure

- Step 1** Enter global configuration mode.
- configure terminal**
- Step 2** (Optional) Display information about XML server settings and active XML server sessions. You can find session numbers in the command output.
- show xml server status**
- Step 3** Validate XML documents for the specified server session.
- xml server validate all**
- Step 4** Terminate the specified XML server session.
- xml server terminate *session***
- Step 5** (Optional) Disable the SSH server so that you can generate keys.
- no feature ssh**
- Step 6** Enable the SSH server. (The default is enabled.)
- feature ssh**
- Step 7** (Optional) Display the status of the SSH server.
- show ssh server**

- Step 8** Set the number of XML server sessions allowed.
xml server max-session *sessions*
 The range is from 1 through 8. The default is 8.
- Step 9** Set the number of seconds after which an XML server session is terminated.
xml server timeout *seconds*
 The range is from 1 through 1200. The default is 1200 seconds.
- Step 10** (Optional) Display information about the XML server settings and active XML server sessions.
show xml server status
- Step 11** (Optional) Saves the running configuration to the startup configuration.
copy running-config startup-config
-

Example

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

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

Starting an SSHv2 Session

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

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

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



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

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

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

Sending a Hello Message

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

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



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

Hello Message from a Server

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

Hello Message from a Client

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

Obtaining XML Schema Definition (XSD) Files

Procedure

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

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

Sending an XML Document to the XML Server

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

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

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

Creating NETCONF XML Instances

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

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

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

NETCONF XML Framework Context

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



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

RPC Request Tag

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

- NETCONF namespace declaration—The `<rpc>` and NETCONF tags that are defined in the `urn:ietf:params:xml:ns:netconf:base:1.0` namespace, are present in the `netconf.xsd` schema file.
- Device namespace declaration—Device tags encapsulated by the `<rpc>` and NETCONF tags are defined in other namespaces. Device namespaces are feature-oriented. Cisco NX-OS feature tags are defined in different namespaces. RPC Request Tag `<rpc>` is an example that uses the NFCLI feature. It declares that the device namespace is `xmlns=http://www.cisco.com/nxos:1.0:nfcli`. `nfcli.xsd` contains this namespace definition. For more information, see [Obtaining XML Schema Definition \(XSD\) Files, on page 249](#).

Examples

RPC Request Tag `<rpc>`

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

Configuration Request

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

```

    </configure>
  </nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>

```



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

NETCONF Operations Tags

NETCONF provides the following configuration operations:

Table 16: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 260
commit	Sets the running configuration to the current contents of candidate configuration.	NETCONF Commit Instance: Candidate Configuration Capability, on page 265
confirmed-commit	Provides the parameters to commit the configuration for a specified time. If a commit operation does not follow this operation within the confirm-timeout period, the configuration is reverted to the state before the confirmed-commit operation.	NETCONF Confirmed Commit Instance, on page 265
copy-config	Copies the contents of the source configuration datastore to the target datastore.	NETCONF Copy Config Instance, on page 261
delete-config	Operation not supported.	—
edit-config	Configures the features in the running configuration of the device. You use this operation for configuration commands.	NETCONF Edit Config Instance, on page 262 NETCONF Rollback-On-Error Instance, on page 266
get	Receives configuration information from a device. You use this operation for show commands. The source of the data is the running configuration.	Creating NETCONF XML Instances, on page 250

NETCONF Operation	Description	Example
get-config	Retrieves all or part of a configuration.	Creating NETCONF XML Instances, on page 250
kill-session	Closes the specified XML server session. You cannot close your own session.	NETCONF Kill Session Instance, on page 261
lock	Allows a client to lock the configuration system of a device.	NETCONF Lock Instance, on page 264
unlock	Releases the configuration lock that the session issued.	NETCONF Unlock Instance, on page 265
validate	Checks the configuration of a candidate for syntactical and semantic errors before applying the configuration to a device.	NETCONF Validate Capability Instance, on page 266

Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See [Obtaining XML Schema Definition \(XSD\) Files, on page 249](#).

Using this schema, it is possible to build an XML instance. The relevant portions of the nfcli.xsd schema file that was used to build the NETCONF instances. See [Creating NETCONF XML Instances, on page 250](#).

show xml Device Tags

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

Server Status Device Tags

```
<xs:complexType name="xml_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>xml agent</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
  <xs:annotation>
```

```

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

```

Device Tag Response

```

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

```



Note The `__XML__OPT_Cmd_show_xml__readonly__` tag is optional. This tag represents the response. For more information on responses, see [RPC Response Tag, on page 259](#).

You can use the | XML option to find the tags that you can use to execute a <get> operation. The following is an example of the | XML option. This example shows you that the namespace-defining tag to execute operations on this device is `http://www.cisco.com/nxos:1.0:nfcli`, and that the `nfcli.xsd` file can be used to build requests.

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

XML Example

```

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

```

```

<server>
<status>
<__XML_OPT_Cmd_show_xml__readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>
<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML_OPT_Cmd_show_xml__readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

Extended NETCONF Operations

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

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

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

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

Table 17: Operation Tags

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

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

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

Configuration CLI Commands Sent Through the <exec-command>

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

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

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

The following is the response to a send operation:

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

Show CLI Commands Sent Through the <exec-command>

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

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

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0"
  xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
<nf:data>
<mod:show>
<mod:interface>
<mod: __XML__OPT_Cmd_show_interface_brief__readonly__>
<mod: __readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
```



```

<mod:ip_addr>192.0.2.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML_OPT_Cmd_show_interface_brief__readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

Failed Configuration

```

<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 192.0.2.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

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

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

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

Combination of configure and show Commands

```

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

```

show CLI Commands Sent Through the <exec-command>

```

<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

NETCONF Replies

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

RPC Response Tag

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

RPC Response Tag <rpc-reply>

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

RPC Response Elements

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

Table 18: RPC Response Elements

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

Interpreting the Tags Encapsulated in the data Tag

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

RPC Reply Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:iETF:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>
<show>
<interface>
<_XML_OPT_Cmd_show_interface_brief__readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
```

```

<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief__readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```



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

Information About Example XML Instances

Example XML Instances

This section provides examples of the following XML instances:

- [NETCONF Close Session Instance, on page 260](#)
- [NETCONF Kill Session Instance, on page 261](#)
- [NETCONF Copy Config Instance, on page 261](#)
- [NETCONF Edit Config Instance, on page 262](#)
- [NETCONF Get Config Instance, on page 263](#)
- [NETCONF Lock Instance, on page 264](#)
- [NETCONF Unlock Instance, on page 265](#)
- [NETCONF Commit Instance: Candidate Configuration Capability, on page 265](#)
- [NETCONF Confirmed Commit Instance, on page 265](#)
- [NETCONF Rollback-On-Error Instance, on page 266](#)
- [NETCONF Validate Capability Instance, on page 266](#)

NETCONF Close Session Instance

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

Close Session Request

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

Close Session Response

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

NETCONF Kill Session Instance

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

Kill Session Request

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

Kill Session Response

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

NETCONF Copy Config Instance

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

Copy Config Request

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

Copy Config Response

```

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

```

NETCONF Edit Config Instance

The following examples show the use of NETCONF edit config:

Edit Config Request

```

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

```

Edit Config Response

```

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

```

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

- create
- merge
- delete

Edit Config: Delete Operation Request

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

```

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

```

Response to Edit Config: Delete Operation

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

```

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

```

NETCONF Get Config Instance

The following examples show the use of NETCONF get config:

Get Config Request to Retrieve the Entire Subtree

```

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

```

Get Config Response with Results of a Query

```

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

```

```

<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>

```

NETCONF Lock Instance

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

Lock Request

```

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

```

Response to a Successful Acquisition of Lock

```

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

```

Response to an Unsuccessful Attempt to Acquire Lock

```

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

```


NETCONF Unlock Instance

The following examples show the use of NETCONF unlock:

Unlock Request

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

Response to an Unlock Request

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

NETCONF Commit Instance: Candidate Configuration Capability

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

Commit Operation

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

Commit Reply

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

NETCONF Confirmed Commit Instance

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

Confirmed Commit Request

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

```
</commit>
</rpc>]]>]]>
```

Confirmed Commit Response

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

NETCONF Rollback-On-Error Instance

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

Rollback-On-Error Capability

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

Rollback-On-Error Response

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

NETCONF Validate Capability Instance

The following examples show the use of NETCONF validate capability. The string `urn:ietf:params:netconf:capability:validate:1.0` identifies the NETCONF validate capability.

Validate Request

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

Response to Validate Request

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

Additional References

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

RFCs

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



APPENDIX **A**

Streaming Telemetry Sources

This chapter contains the following topics:

- [About Streaming Telemetry, on page 269](#)
- [Guidelines and Limitations, on page 269](#)
- [Data Available for Telemetry, on page 269](#)

About Streaming Telemetry

The streaming telemetry feature of the Cisco Nexus 3400-S switches continuously streams data out of the network and notifies the client, providing near-real-time access to monitoring data.

Guidelines and Limitations

Following are the guidelines and limitations for streaming telemetry:

- The telemetry feature is available in Cisco Nexus 3400-S switches.
- Switches with less than 8 GB of memory do not support telemetry.

Data Available for Telemetry

For each component group, the distinguished names (DNs) in the appendix of the [NX-API DME Model Reference](#) can provide the listed properties as data for telemetry.

