



Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 10.5(x)

First Published: 2024-07-26

Last Modified: 2024-11-27

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 REFERENCED IN THIS DOCUMENTATION ARE SUBJECT TO CHANGE WITHOUT NOTICE. EXCEPT AS MAY OTHERWISE BE AGREED BY CISCO IN WRITING, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENTATION ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

The Cisco End User License Agreement and any supplemental license terms govern your use of any Cisco software, including this product documentation, and are located at <https://www.cisco.com/c/en/us/about/legal/cloud-and-software/software-terms.html>. Cisco product warranty information is available at <https://www.cisco.com/c/en/us/products/warranty-listing.html>. US Federal Communications Commission Notices are found here <https://www.cisco.com/c/en/us/products/us-fcc-notice.html>.

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 products and features described herein as in development or available at a future date remain in varying stages of development and will be offered on a when-and if-available basis. Any such product or feature roadmaps are subject to change at the sole discretion of Cisco and Cisco will have no liability for delay in the delivery or failure to deliver any products or feature roadmap items that may be set forth in this document.

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.

The documentation set for this product strives to use bias-free language. For the purposes of this documentation set, bias-free is defined as language that does not imply discrimination based on age, disability, gender, racial identity, ethnic identity, sexual orientation, socioeconomic status, and intersectionality. Exceptions may be present in the documentation due to language that is hardcoded in the user interfaces of the product software, language used based on RFP documentation, or language that is used by a referenced third-party product.

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/c/en/us/about/legal/trademarks.html>. 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)

© 2024 Cisco Systems, Inc. All rights reserved.



CONTENTS

PREFACE

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

CHAPTER 1

New and Changed Information	1
New and Changed Information	1

CHAPTER 2

Overview	3
Licensing Requirements	3
Supported Platforms	3

CHAPTER 3

Bash	5
About Bash	5
Accessing Bash	5
Escalate Privileges to Root	6
Examples of Bash Commands	7
Displaying System Statistics	7
Running Bash from CLI	8
Running Python from Bash	8
Copy Through Kstack	9

CHAPTER 4

Guest Shell	11
About the Guest Shell	11

- Guidelines and Limitations 12
- Guidelines and Limitations for Guestshell 16
- Guidelines and Limitations for Guestshell 22
- Accessing the Guest Shell 28
- Resources Used for the Guest Shell 29
- Capabilities in the Guestshell 30
 - NX-OS CLI in the Guest Shell 30
 - Network Access in Guest Shell 31
 - Access to Bootflash in Guest Shell 33
 - Python in Guest Shell 33
 - Python 3 in Guest Shell versions up to 2.10 (CentOS 7) 34
 - Python 3 in Guest Shell versions up to 2.10 (CentOS 7) 36
 - Installing RPMs in the Guest Shell 39
- Security Posture for Virtual ServicesGuest Shell 40
 - Digitally Signed Application Packages 41
 - Kernel Vulnerability Patches 41
 - ASLR and X-Space Support 41
 - Namespace Isolation 42
 - Namespace Isolation 42
 - Root-User Restrictions 43
 - Resource Management 44
- Guest File System Access Restrictions 44
- Managing the Guest Shell 44
 - Disabling the Guest Shell 48
 - Destroying the Guest Shell 49
 - Enabling the Guest Shell 49
 - Replicating the Guest Shell 51
 - Exporting Guest Shell rootfs 51
 - Importing Guest Shell rootfs 51
 - Importing YAML File 53
 - show guestshell Command 56
- Verifying Virtual Service and Guest Shell Information 57
- Persistently Starting Your Application From the Guest Shell 58
- Procedure for Persistently Starting Your Application from the Guest Shell 59

An Example Application in the Guest Shell 59

Troubleshooting Guest Shell Issues 60

CHAPTER 5

Python API 63

Information About the Python API 63

Using Python 63

Cisco Python Package 64

Using the CLI Command APIs 65

Invoking the Python Interpreter from the CLI 66

Display Formats 67

Non-Interactive Python 69

Running Scripts with Embedded Event Manager 70

Python Integration with Cisco NX-OS Network Interfaces 71

Cisco NX-OS Security with Python 72

Examples of Security and User Authority 72

Example of Running Script with Scheduler 72

CHAPTER 6

Python API 73

About the Python API 73

Using Python 73

Cisco Python Package 74

Using the CLI Command APIs 75

Invoking the Python Interpreter from the CLI 76

Display Formats 77

Non-Interactive Python 79

Running Scripts with Embedded Event Manager 80

Python Integration with Cisco NX-OS Network Interfaces 81

Cisco NX-OS Security with Python 81

Examples of Security and User Authority 81

Example of Running Script with Scheduler 83

CHAPTER 7

Scripting with Tcl 85

About Tcl 85

Tclsh Command Help 85

- Telsh Command History 86
- Telsh Tab Completion 86
- Telsh CLI Command 86
- Telsh Command Separation 86
- Tel Variables 87
- Telquit 87
- Telsh Security 87
- Running the Telsh Command 87
- Navigating Cisco NX-OS Modes from the Telsh Command 88
- Tel References 90

CHAPTER 8

Ansible 91

- Prerequisites 91
- About Ansible 91
- Cisco Ansible Module 91

CHAPTER 9

Puppet Agent 93

- About Puppet 93
- Prerequisites 94
- Puppet Agent NX-OS Environment 94
- ciscopuppet Module 94

CHAPTER 10

SaltStack 97

- About SaltStack 97
 - About NX-OS and SaltStack 98
- Guidelines and Limitations 98
- Cisco NX-OS Environment for SaltStack 98
- Enabling NX-API for SaltStack 99
- Installing SaltStack for NX-OS 99

CHAPTER 11

Using Chef Client with Cisco NX-OS 101

- About Chef 101
- Prerequisites 101
- Chef Client NX-OS Environment 102

cisco-cookbook 102

CHAPTER 12

Using Docker with Cisco NX-OS 105

About Docker with Cisco NX-OS 105

Guidelines and Limitations 105

Prerequisites for Setting Up Docker Containers Within Cisco NX-OS 106

Starting the Docker Daemon 106

Configure Docker to Start Automatically 107

Starting Docker Containers: Host Networking Model 108

Starting Docker Containers: Bridged Networking Model 109

Mounting the bootflash and volatile Partitions in the Docker Container 110

Enabling Docker Daemon Persistence on Enhanced ISSU Switchover 110

Enabling Docker Daemon Persistence on the Cisco Nexus Platform Switches Switchover 111

Resizing the Docker Storage Backend 112

Stopping the Docker Daemon 114

Docker Container Security 115

 Securing Docker Containers With User namespace Isolation 115

 Moving the cgroup Partition 116

Adding Nodes to a Kubernetes Cluster 116

Docker Troubleshooting 119

 Docker Fails to Start 119

 Docker Fails to Start Due to Insufficient Storage 119

 Failure to Pull Images from Docker Hub (509 Certificate Expiration Error Message) 120

 Failure to Pull Images from Docker Hub (Client Timeout Error Message) 120

 Docker Daemon or Containers Not Running On Switch Reload or Switchover 121

 Resizing of Docker Storage Backend Fails 121

 Docker Container Doesn't Receive Incoming Traffic On a Port 122

 Unable to See Data Port And/Or Management Interfaces in Docker Container 122

 General Troubleshooting Tips 122

CHAPTER 13

NX-SDK 123

About the NX-SDK 123

 Considerations for Go Bindings 124

About On-Box (Local) Applications 124

- Default Docker Images 124
- Guidelines and Limitations for NX-SDK 125
- Off-Box (Remote) Applications 126
 - About NX-SDK 2.0 126
 - About Remote Applications 126
 - NX-SDK Security 127
 - Security Profiles for NX SDK 2.0 127

CHAPTER 14

NX-API 129

- About NX-API 129
 - Feature NX-API 129
 - Transport 130
 - Message Format 130
 - Security 130
- Using NX-API 131
 - NX-API Management Commands 134
 - Working With Interactive Commands Using NX-API 136
 - NX-API Request Elements 136
 - NX-API Response Elements 146
 - About JSON (JavaScript Object Notation) 147
 - Notes about JSON 147
 - CLI Execution 147
 - XML and JSON Supported Commands 148
 - Examples of XML and JSON Output 148
 - JSON-RPC, JSON, and XML Supported Commands 156
 - Examples of XML and JSON Output 157

CHAPTER 15

NX-API Response Codes 161

- Table of NX-API Response Codes 161

CHAPTER 16

NX-API Developer Sandbox 165

- NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2) 165
 - About the NX-API Developer Sandbox 165
 - Guidelines and Limitations 166

	Configuring the Message Format and Command Type	166
	Using the Developer Sandbox	168
	Using the Developer Sandbox to Convert CLI Commands to Payloads	168
	NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later	170
	About the NX-API Developer Sandbox	170
	Guidelines and Limitations	171
	Configuring the Message Format and Input Type	173
	Using the Developer Sandbox	176
	Using the Developer Sandbox to Convert CLI Commands to REST Payloads	177
	Using the Developer Sandbox to Convert from REST Payloads to CLI Commands	179
	Using the Developer Sandbox to Convert from RESTCONF to json or XML	184
<hr/>		
CHAPTER 17	XML Support for ABM and LM in N3500	187
	XML Support for ABM and LM in N3500	187
<hr/>		
CHAPTER 18	Converting CLI Commands to Network Configuration Format	195
	Information About XMLIN	195
	Licensing Requirements for XMLIN	195
	Installing and Using the XMLIN Tool	196
	Converting Show Command Output to XML	196
	Configuration Examples for XMLIN	197
<hr/>		
CHAPTER 19	OpenConfig YANG	201
	About OpenConfig YANG	201
	Guidelines and Limitations for OpenConfig YANG	201
	Understanding Deletion of BGP Routing Instance	209
	Verifying YANG	211
	Enabling OpenConfig Support	211
<hr/>		
CHAPTER 20	XML Management Interface	213
	About the XML Management Interface	213
	About the XML Management Interface	213
	NETCONF Layers	213
	SSH xmlagent	214

- Licensing Requirements for the XML Management Interface 214
- Prerequisites to Using the XML Management Interface 215
- Using the XML Management Interface 215
 - Configuring SSH and the XML Server Options 215
 - Starting an SSH Session 215
 - Sending the Hello Message 216
 - Obtaining the XSD Files 216
 - Sending an XML Document to the XML Server 217
 - Creating NETCONF XML Instances 217
 - RPC Request Tag rpc 218
 - NETCONF Operations Tags 219
 - Device Tags 220
 - Extended NETCONF Operations 222
 - NETCONF Replies 225
 - RPC Response Tag 226
 - Interpreting Tags Encapsulated in the Data Tag 226
- Information About Example XML Instances 227
 - Example XML Instances 227
 - NETCONF Close Session Instance 227
 - NETCONF Kill-session Instance 228
 - NETCONF copy-config Instance 228
 - NETCONF edit-config Instance 228
 - NETCONF get-config Instance 230
 - NETCONF Lock Instance 230
 - NETCONF unlock Instance 231
 - NETCONF Commit Instance - Candidate Configuration Capability 232
 - NETCONF Confirmed-commit Instance 232
 - NETCONF rollback-on-error Instance 232
 - NETCONF validate Capability Instance 233
- Additional References 233

PART I

Model-Driven Programmability 235

CHAPTER 21

Managing Components 237

About the Component RPM Packages	237
Preparing For Installation	239
Downloading Components from the Cisco Artifactory	240
Installing RPM Packages	241
Installing the Programmable Interface Base And Common Model Component RPM Packages	241

CHAPTER 22**Converting CLI Commands to Network Configuration Format 243**

Information About XMLIN	243
Licensing Requirements for XMLIN	243
Installing and Using the XMLIN Tool	244
Converting Show Command Output to XML	244
Configuration Examples for XMLIN	245

CHAPTER 23**gNMI - gRPC Network Management Interface 249**

About gNMI	249
gNMI RPC and SUBSCRIBE	250
Guidelines and Limitations for gNMI	251
Configuring gNMI	254
gNMI - gRPC Network Management Interface	255
Configuring Server Certificate	256
Generating Key/Certificate Examples	257
Generating Key/Certificate Examples	258
Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3(3) and Later	258
Enabling gNMI	260
Verifying gNMI	261
gRPC Client-Certificate-Authentication	267
Generating New Client Root CA Certificates	267
Configuring the Generated Root CA Certificates on NX-OS Device	268
Associating Trustpoints to gRPC	268
Validating the Certificate Details	269
Verifying the Connection using Client Certificate Authentication for any gNMI Clients	270
Clients	270
Sample DME Subscription - JSON Encoding	271
Sample DME Subscription - PROTO Encoding	272

Capabilities	273
About Capabilities	273
Guidelines and Limitations for Capabilities	273
Example Client Output for Capabilities	274
Get	277
About Get	277
Guidelines and Limitations for Get	277
Set	278
About Set	278
Guidelines and Limitations for Set	278
Subscribe	279
Guidelines and Limitations for Subscribe	279
gNMI Payload	280
Streaming Syslog	282
About Streaming Syslog for gNMI	282
Guidelines and Limitations for Streaming Syslog - gNMI	283
Syslog Native YANG Model	283
Subscribe Request Example	284
Sample PROTO Output	285
Sample JSON Output	287
Troubleshooting	288
Gathering TM-Trace Logs	288
Gathering MTX-Internal Logs	289

CHAPTER 24	gNOI-gRPC Network Operations Interface	293
	About gNOI	293
	Supported gNOI RPCs	293
	System Proto	294
	OS Proto	295
	Cert Proto	296
	File Proto	296
	gNOI Factory Reset	297
	Guidelines and Limitations	298
	Verifying gNOI	298

CHAPTER 25**Model Driven Telemetry 301**

- About Telemetry **301**
 - Telemetry Components and Process **301**
 - High Availability of the Telemetry Process **303**
- Licensing Requirements for Telemetry **303**
- Installing and Upgrading Telemetry **303**
- Guidelines and Limitations for Model Driven Telemetry **304**
- Configuring Telemetry Using the CLI **311**
 - Configuring Telemetry Using the NX-OS CLI **311**
 - Configuring Cadence for YANG Paths **316**
 - Configuration Examples for Telemetry Using the CLI **318**
 - Displaying Telemetry Configuration and Statistics **322**
 - Displaying Telemetry Log and Trace Information **330**
- Configuring Telemetry Using the NX-API **332**
 - Configuring Telemetry Using the NX-API **332**
 - Configuration Example for Telemetry Using the NX-API **342**
 - Telemetry Model in the DME **345**
- Telemetry Path Labels **346**
 - About Telemetry Path Labels **346**
 - Polling for Data or Receiving Events **347**
 - Guidelines and Limitations for Path Labels **347**
 - Configuring the Interface Path to Poll for Data or Events **347**
 - Configuring the Interface Path for Non-Zero Counters **349**
 - Configuring the Interface Path for Operational Speeds **351**
 - Configuring the Interface Path with Multiple Queries **353**
 - Configuring the Environment Path to Poll for Data or Events **354**
 - Configuring the Resources Path for Poll for Events or Data **356**
 - Configuring the VXLAN Path to Poll for Events or Data **358**
 - Verifying the Path Label Configuration **359**
 - Displaying Path Label Information **360**
- Native Data Source Paths **363**
 - About Native Data Source Paths **363**
 - Telemetry Data Streamed for Native Data Source Paths **363**

Guidelines and Limitations for Native Data Source Path	365
Configuring the Native Data Source Path for Routing Information	366
Configuring the Native Data Source Path for MAC Information	368
Configuring the Native Data Source Path for All MAC Information	370
Configuring the Native Data Path for IP Adjacencies	372
Displaying Native Data Source Path Information	374
Streaming Syslog	375
About Streaming Syslog for Telemetry	375
Configuring the Native Data Source Path for Routing Information	375
Telemetry Data Streamed for Syslog Path	378
Sample JSON Output	379
Sample KVGPB Output	379
Additional References	382
Related Documents	382

APPENDIX A

Streaming Telemetry Sources	383
About Streaming Telemetry	383
Guidelines and Limitations	383
Data Available for Telemetry	383



Preface

This preface includes the following sections:

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

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 results 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 DevNet](#).
- 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

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

New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 10.5(x)* and where they are documented.

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
NA	No new features added for this release.	10.5(1)F	NA



CHAPTER 2

Overview

- [Licensing Requirements, on page 3](#)
- [Supported Platforms, on page 3](#)

Licensing Requirements

For a complete explanation of Cisco NX-OS licensing recommendations and how to obtain and apply licenses, see the [Cisco NX-OS Licensing Guide](#) and the [Cisco NX-OS Licensing Options Guide](#).

Supported Platforms

Starting with Cisco NX-OS release 7.0(3)I7(1), use the [Nexus Switch Platform Support Matrix](#) to know from which Cisco NX-OS releases various Cisco Nexus 9000 and 3000 switches support a selected feature.



CHAPTER 3

Bash

- [About Bash, on page 5](#)
- [Accessing Bash, on page 5](#)
- [Escalate Privileges to Root, on page 6](#)
- [Examples of Bash Commands, on page 7](#)
- [Copy Through Kstack, on page 9](#)

About Bash

In addition to the Cisco NX-OS CLI, Cisco Nexus 3500 platform switches support access to the Bourne-Again SHell (Bash). Bash interprets commands that you enter or commands that are read from a shell script. Using Bash enables access to the underlying Linux system on the device and to manage the system.

Accessing Bash

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

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

```
switch# show role name dev-ops

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

```
switch# show role name network-admin

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

```

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

```

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

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

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

```

switch# configure terminal
switch(config)# feature bash-shell

switch# run bash
Linux# whoami
admin
Linux# pwd
/bootflash/home/admin
Linux#

```



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

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

```
run bash whoami
```

Escalate Privileges to Root

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

The following are guidelines for escalating privileges:

- 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.

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

```

switch# run bash
Linux# sudo su root

```

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

- ```

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

```

Password:



```
Linux# whoami
root
Linux# exit
exit
```

## Examples of Bash Commands

This section contains examples of Bash commands and output.

### Displaying System Statistics

The following example shows how to display system statistics:

```
switch# run bash
Linux# cat /proc/meminfo
MemTotal: 3795100 kB
MemFree: 1472680 kB
Buffers: 136 kB
Cached: 1100116 kB
ShmFS: 1100116 kB
Allowed: 948775 Pages
Free: 368170 Pages
Available: 371677 Pages
SwapCached: 0 kB
Active: 1198872 kB
Inactive: 789764 kB
SwapTotal: 0 kB
SwapFree: 0 kB
Dirty: 0 kB
Writeback: 0 kB
AnonPages: 888272 kB
Mapped: 144044 kB
Slab: 148836 kB
SReclaimable: 13892 kB
SUnreclaim: 134944 kB
PageTables: 28724 kB
NFS_Unstable: 0 kB
Bounce: 0 kB
WritebackTmp: 0 kB
CommitLimit: 1897548 kB
Committed_AS: 19984932 kB
VmallocTotal: 34359738367 kB
VmallocUsed: 215620 kB
VmallocChunk: 34359522555 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 2048 kB
DirectMap4k: 40960 kB
DirectMap2M: 4190208 kB
Linux#
```

## Running Bash from CLI

The following example shows how to run a bash command from the CLI with the `run bash <command>` command:

```
switch# run bash ps -el
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
4 S 0 1 0 0 80 0 - 497 select ? 00:00:08 init
5 S 0 2 0 0 75 -5 - 0 kthrea ? 00:00:00 kthreadd
1 S 0 3 2 0 -40 - - 0 migrat ? 00:00:00 migration/0
1 S 0 4 2 0 75 -5 - 0 ksofti ? 00:00:01 ksoftirqd/0
5 S 0 5 2 0 58 - - 0 watchd ? 00:00:00 watchdog/0
1 S 0 6 2 0 -40 - - 0 migrat ? 00:00:00 migration/1
1 S 0 7 2 0 75 -5 - 0 ksofti ? 00:00:00 ksoftirqd/1
5 S 0 8 2 0 58 - - 0 watchd ? 00:00:00 watchdog/1
1 S 0 9 2 0 -40 - - 0 migrat ? 00:00:00 migration/2
1 S 0 10 2 0 75 -5 - 0 ksofti ? 00:00:00 ksoftirqd/2
5 S 0 11 2 0 58 - - 0 watchd ? 00:00:00 watchdog/2
1 S 0 12 2 0 -40 - - 0 migrat ? 00:00:00 migration/3
1 S 0 13 2 0 75 -5 - 0 ksofti ? 00:00:00 ksoftirqd/3
5 S 0 14 2 0 58 - - 0 watchd ? 00:00:00 watchdog/3

...

4 S 0 8864 1 0 80 0 - 2249 wait ttyS0 00:00:00 login
4 S 2002 28073 8864 0 80 0 - 69158 select ttyS0 00:00:00 vsh
4 R 0 28264 3782 0 80 0 - 54790 select ? 00:00:00 in.dcos-telnet
4 S 0 28265 28264 0 80 0 - 2247 wait pts/0 00:00:00 login
4 S 2002 28266 28265 0 80 0 - 69175 wait pts/0 00:00:00 vsh
1 S 2002 28413 28266 0 80 0 - 69175 wait pts/0 00:00:00 vsh
0 R 2002 28414 28413 0 80 0 - 887 - pts/0 00:00:00 ps

switch#
```

## Running Python from Bash

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

```
switch# run bash
Linux# python
Python 2.7.5 (default, May 16 2014, 10:58:01)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NxOS lib!
>>>
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

!Command: show running-config interface Ethernet1/3
!Time: Thu Aug 21 23:32:25 2014

version 6.0(2)U4(1)

interface Ethernet1/3
 no switchport
```

```
vrf member myvrf
```

```
>>>
```

## Copy Through Kstack

In Cisco NX-OS release 9.3(1) and later, file copy operations have the option of running through a different network stack by using the **use-kstack** option. Copying files through **use-kstack** enables faster copy times. This option can be beneficial when copying files from remote servers that are multiple hops from the switch. The **use-kstack** option work with copying files from, and to, the switch though standard file copy features, such as **scp** and **sftp**.



---

**Note** The **use-kstack** option does not work when the switch is running the FIPS mode feature. If the switch has FIPS mode that is enabled, the copy operation is still successful, but through the default copy method.

---

To copy through **use-kstack**, append the argument to the end of an NX-OS **copy** command. Some examples:

```
switch-1# copy scp://test@10.1.1.1/image.bin . vrf management use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf management
use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin . use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf default
use-kstack
switch-1#
```

The **use-kstack** option is supported for all NX-OS **copy** commands and file systems. The option is OpenSSL (Secure Copy) certified.





## CHAPTER 4

# Guest Shell

---

- [About the Guest Shell, on page 11](#)
- [Guidelines and Limitations, on page 12](#)
- [Guidelines and Limitations for Guestshell, on page 16](#)
- [Guidelines and Limitations for Guestshell, on page 22](#)
- [Accessing the Guest Shell, on page 28](#)
- [Resources Used for the Guest Shell, on page 29](#)
- [Capabilities in the Guestshell, on page 30](#)
- [Security Posture for Virtual ServicesGuest Shell, on page 40](#)
- [Guest File System Access Restrictions , on page 44](#)
- [Managing the Guest Shell, on page 44](#)
- [Verifying Virtual Service and Guest Shell Information, on page 57](#)
- [Persistently Starting Your Application From the Guest Shell, on page 58](#)
- [Procedure for Persistently Starting Your Application from the Guest Shell, on page 59](#)
- [An Example Application in the Guest Shell, on page 59](#)
- [Troubleshooting Guest Shell Issues, on page 60](#)

## About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, switches 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 the switch's bootflash.
- Access to the switch's volatile tmpfs.
- Access to the switch's CLI.
- Access to the switch's host file system.
- 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 will appear in the virtual-service show command output.




---

**Note** By default, the Guest Shell occupies approximately 5 MB of RAM and 200 MB of bootflash when enabled. Beginning with Cisco NX-OS Release 7.0(3)I2(1) the Guest Shell occupies approximately 35 MB of RAM. Use the **guestshell destroy** command to reclaim resources if the Guest Shell is not used.

---




---

**Note** Beginning with Cisco NX-OS 7.0(3)F3(1)NX-OS 7.0(3)I7(1), the Guest Shell is supported on the Cisco Nexus 95083500 switch.

---

## Guidelines and Limitations

The Guest Shell has the following guideline 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 an 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 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.
- `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. `sshd` 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 prior to 2.2 (0.2) with Cisco Nexus release 7.0(3)I5(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 and/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 vegas-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 port 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 starting at 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/ssh-mgmt.service` and `/etc/ssh/ssh-mgmt_config`. The files should have the following configurations:
 

```
-rw-r--r-- 1 root root 394 Apr 7 14:21 /usr/lib/systemd/system/ssh-mgmt.service
-rw----- 1 root root 4478 Apr 7 14:22 /etc/ssh/ssh-mgmt_config
```
2. Copy the Unit and Service contents from the `/usr/lib/systemd/system/ssh.service` file to `ssh-mgmt.service`.
3. Edit the `ssh-mgmt.service` file to match the following:
 

```
[Unit]
Description=OpenSSH server daemon
After=network.target ssh-keygen.service
Wants=ssh-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/ssh
ExecStartPre=/usr/sbin/ssh-keygen
ExecStart=/sbin/ip netns exec management /usr/sbin/ssh -f /etc/ssh/ssh-mgmt_config
-D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target
```
4. Copy the contents of `/etc/ssh/ssh-config` to `/etc/ssh/ssh-mgmt_config`. Modify the ListenAddress IP and port as necessary.
 

```
Port 2222
ListenAddress 10.122.84.34
```
5. Start the `systemctl` daemon using the following commands:
 

```
sudo systemctl daemon-reload
sudo systemctl start ssh-mgmt.service
sudo systemctl status ssh-mgmt.service -l
```
6. (optional) Check the configuration.
 

```
ss -tnldp | grep 2222
```
7. SSH into Guest Shell:
 

```
ssh -p 2222 guestshell@10.122.84.34
```
8. Save the configuration across multiple Guest Shell or switch reboots.
 

```
sudo systemctl enable ssh-mgmt.service
```
9. For passwordless SSH/SCP and remote execution, generate the public and private keys for the user ID you want to use 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
```

10. 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/*
```

11. 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
```

## Guidelines and Limitations for Guestshell

### Common Guidelines Across All Releases



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

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

- Guest Shell is not supported on 3500 models with 4GB of memory (3524, 3548, 3524-X, 3548-X). It is supported on the platforms with higher memory, such as -XL.
- If you are running a third-party DHCPD server in Guestshell, there might be issues with offers reaching the client if used along with SVI. A possible workaround is to use broadcast responses.
- Use the `run guestshell` CLI command to access the Guestshell on the switch: The `run guestshell` command parallels the `run bash` command that is used to access the host shell. This command allows

you to access the Guestshell and get a Bash prompt or run a command within the context of the Guestshell. The command uses password-less SSH to an available port on the localhost in the default network namespace.

- When routes are being exchanged between two different VRFs in NXOS (either statically or dynamically), the routing table for the corresponding VRF / Namespace does not get populated with the "shared route" in the guestshell container.
- The `sshd` utility can secure the pre-configured SSH access into the Guestshell by listening on `localhost` to avoid connection attempts from outside the network. The `sshd` has the following features:
  - It is configured for key-based authentication without fallback to passwords.
  - Only `root` can read keys use to access the Guestshell after Guestshell restarts.
  - Only `root` can read the file that contains the key on the host to prevent a nonprivileged user with host Bash access from being able to use the key to connect to the Guestshell. Network-admin users may start another instance of `sshd` in the Guestshell to allow remote access directly into the Guestshell, but any user that logs into the Guestshell is also given network-admin privilege.




---

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

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

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

---

- Installing the Cisco NX-OS software release on a fresh out-of-the-box switch will automatically enable the Guestshell. Subsequent upgrades to the switch software will not automatically upgrade Guestshell.
- Guestshell releases increment the major number when distributions or distribution versions change.
- Guestshell for NX-OS can access front-panel ports as first-class Linux interfaces.
- Guestshell for NX-OS can access Command shell through `dohost` using local Unix socket to NX-API.
  1. Guestshell for NX-OS: Access to NX-API socket is allowed only for root/admin user privilege from 9.3(8) and later.
  2. Guestshell for NX-OS: Access to NX-OS filesystem only as root/admin user in 9.3(8) and later.
- Guestshell releases increment the minor number when CVEs have been addressed. The Guestshell updates CVEs only when CentOS makes them publicly available.
- Cisco recommends using **`dnf 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 Guestshell rootfs. Any customizations and software package installations would then need to be performed again within the context of this new Guestshell rootfs.
- Setting Nexus clock from bash shell is not supported.

### CentOS end of life and impact on Guestshell

Guestshell is an **LXC container based on CentOS environment**. As per updates in the open source community, CentOS 8 Project is reaching end of support by December 2021. The CentOS 7 project is to continue through and is targeted to reach end of support by June 2024. Due to this long term support for CentOS 7, the latest Cisco NX-OS software 10.2.x is packaged with Guestshell 2.11 (CentOS 7 based). This replaces Guestshell 3.0 (CentOS 8) which is the default environment in 10.1.x release.

#### Guestshell 2.11

Beginning with Cisco NX-OS release 10.2(1), CentOS 7 is re-introduced as the default Guestshell environment. See section "*CentOS End of Life*" for a detailed explanation on the reasons.

Guestshell 2.11 comes with python2 and python3.6 support. The functionality between Guestshell 2.11 and Guestshell 3.0 remains the same.




---

**Note** The rootfs size of Guestshell 2.11 has increased to approximately 200 MB.

---

#### Guestshell 3.0

Guestshell 3.0 is deprecated and is not available from NX-OS 10.2.x. It is recommended to use Guestshell 2.11. However, the 10.2.x software shall remain compatible with Guestshell 3.0 containers and 3.0 guestshell containers running on 10.1.x. software shall continue to function after upgrade to 10.2.x.




---

**Note** The rootfs size in Guestshell 3.0 is 220 MB versus the 170 MB in Guestshell 2.0.

---

#### Guestshell 4.x

Guestshell 2.x contains Centos 7. End of life for Centos 7 is early 2024. Hence, Guestshell 4.x is a RockyLinux 9 based lxc container that will replace Guestshell 2.x. Guestshell 4.x is available in the following NX-OS releases as downloadable and default options.

| NXOS release        | Guestshell default package version | Guestshell downloadable option applicable |
|---------------------|------------------------------------|-------------------------------------------|
| 9.3(14) and above   | 4.1 or above                       | Not needed                                |
| 10.2(6)             | 2.15                               | 4.0                                       |
| 10.2(7)             | 2.15                               | 4.1                                       |
| 10.2(8) and above   | 4.1 or above                       | Not needed                                |
| 10.3(4)             | 2.15                               | 4.0                                       |
| 10.3(5) and above   | 4.1 or above                       | Not needed                                |
| 10.4(1) and 10.4(2) | 2.15                               | 4.0                                       |
| 10.4(3) and above   | 4.1 or above                       | Not needed                                |

| NXOS release      | Guestshell default package version | Guestshell downloadable option applicable |
|-------------------|------------------------------------|-------------------------------------------|
| 10.5(1) and above | 4.1 or above                       | Not needed                                |



**Note** The rootfs size in Guestshell 4.x is 400 MB versus the 350 MB in Guestshell 2.x. Guestshell 4.x downloadable OVA is backward compactable to all releases running Guestshell 2.x as default.

### Upgrading from Guestshell 1.0 to Guestshell 2.x

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

### Downgrading NX-OS from Jacksonville release Guestshell 3.0

Beginning with Cisco NX-OS release 10.1(1), infrastructure version for Guestshell 3.0 support is increased to 1.11 (check with `show virtual-service` command). Therefore, Guestshell 3.0 OVA cannot be used in previous releases. If used, the **Install all** command will validate version mismatch and throws an error. It is recommended to destroy Guestshell 3.0 before downgrading to previous releases so that Guestshell 3.0 does not come up in previous releases.

### Upgrading from Guestshell 2.x to Guestshell 4.x Downloadable OVA

Guestshell 4.x can be downloaded from Cisco's official software download page and can be installed using command `guestshell upgrade` command.

Following table shows the guest shell releases:

*Table 2: Guest Shell Releases*

| Guest Shell Releases | NX-OS Supported Releases | Python Version(s) Supported |
|----------------------|--------------------------|-----------------------------|
| 2.x                  | 10.4.1                   | python2.7 and python 3.6    |
| 3.0                  | 10.1.x                   | python 3.6                  |
| 4.x                  | 10.4.1                   | python 3.9                  |

### Use below commands to upgrade to Guestshell 4.x:

- Execute command `guestshell enable package <downloaded ova>` when guestshell is not installed.
- Execute command `guestshell upgrade package <downloaded ova>` when guestshell is installed and running.



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

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: Guestshell 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 Guestshell with the **guestshell destroy** command before reloading an older Cisco NX-OS image that does not support the Guestshell.

---

### Pre-Configured SSHD Service

The Guestshell 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 Guestshell 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 Guestshell from the NX-OS CLI might not work.

The following steps instantiate an OpenSSH server within the Guestshell 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 Guestshell sshd service for password-less access uses a randomized port starting at 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`.
3. Edit the `sshd-mgmt.service` file to match the following:
 

```
[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
```
4. 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
```
5. 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
```
6. (Optional) Check the configuration.
 

```
ss -tnldp | grep 2222
```
7. SSH into Guestshell:
 

```
ssh -p 2222 guestshell@10.122.84.34
```
8. Save the configuration across multiple Guestshell or switch reboots.

```
sudo systemctl enable sshd-mgmt.service
```

- For passwordless SSH/SCP and remote execution, generate the public and private keys for the user ID you want to use 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
```

- 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/*
```

- 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 Guestshell 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 Guestshell 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
```

## Guidelines and Limitations for Guestshell

### Common Guidelines Across All Releases




---

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

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

---

- Guest Shell is not supported on 3500 models with 4GB of memory (3524, 3548, 3524-X, 3548-X). It is supported on the platforms with higher memory, such as -XL.



- If you are running a third-party DHCPD server in Guestshell, there might be issues with offers reaching the client if used along with SVI. A possible workaround is to use broadcast responses.
- Use the `run guestshell` CLI command to access the Guestshell on the switch: The `run guestshell` command parallels the `run bash` command that is used to access the host shell. This command allows you to access the Guestshell and get a Bash prompt or run a command within the context of the Guestshell. The command uses password-less SSH to an available port on the localhost in the default network namespace.
- When routes are being exchanged between two different VRFs in NXOS (either statically or dynamically), the routing table for the corresponding VRF / Namespace does not get populated with the "shared route" in the guestshell container.
- The `sshd` utility can secure the pre-configured SSH access into the Guestshell by listening on `localhost` to avoid connection attempts from outside the network. The `sshd` has the following features:
  - It is configured for key-based authentication without fallback to passwords.
  - Only `root` can read keys use to access the Guestshell after Guestshell restarts.
  - Only `root` can read the file that contains the key on the host to prevent a nonprivileged user with host Bash access from being able to use the key to connect to the Guestshell. Network-admin users may start another instance of `sshd` in the Guestshell to allow remote access directly into the Guestshell, but any user that logs into the Guestshell is also given network-admin privilege.




---

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

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

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

---

- Installing the Cisco NX-OS software release on a fresh out-of-the-box switch will automatically enable the Guestshell. Subsequent upgrades to the switch software will not automatically upgrade Guestshell.
- Guestshell releases increment the major number when distributions or distribution versions change.
- Guestshell for NX-OS can access front-panel ports as first-class Linux interfaces.
- Guestshell for NX-OS can access Command shell through `dohost` using local Unix socket to NX-API.
  1. Guestshell for NX-OS: Access to NX-API socket is allowed only for `root/admin` user privilege from 9.3(8) and later.
  2. Guestshell for NX-OS: Access to NX-OS filesystem only as `root/admin` user in 9.3(8) and later.
- Guestshell releases increment the minor number when CVEs have been addressed. The Guestshell updates CVEs only when CentOS makes them publicly available.
- Cisco recommends using **dnf 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 Guestshell rootfs. Any customizations and software package installations would then need to be performed again within the context of this new Guestshell rootfs.

- Setting Nexus clock from bash shell is not supported.

### CentOS end of life and impact on Guestshell

Guestshell is an **LXC container based on CentOS environment**. As per updates in the open source community, CentOS 8 Project is reaching end of support by December 2021. The CentOS 7 project is to continue through and is targeted to reach end of support by June 2024. Due to this long term support for CentOS 7, the latest Cisco NX-OS software 10.2.x is packaged with Guestshell 2.11 (CentOS 7 based). This replaces Guestshell 3.0 (CentOS 8) which is the default environment in 10.1.x release.

### Guestshell 2.11

Beginning with Cisco NX-OS release 10.2(1), CentOS 7 is re-introduced as the default Guestshell environment. See section "*CentOS End of Life*" for a detailed explanation on the reasons.

Guestshell 2.11 comes with python2 and python3.6 support. The functionality between Guestshell 2.11 and Guestshell 3.0 remains the same.




---

**Note** The rootfs size of Guestshell 2.11 has increased to approximately 200 MB.

---

### Guestshell 3.0

Guestshell 3.0 is deprecated and is not available from NX-OS 10.2.x. It is recommended to use Guestshell 2.11. However, the 10.2.x software shall remain compatible with Guestshell 3.0 containers and 3.0 guestshell containers running on 10.1.x. software shall continue to function after upgrade to 10.2.x.




---

**Note** The rootfs size in Guestshell 3.0 is 220 MB versus the 170 MB in Guestshell 2.0.

---

### Guestshell 4.x

Guestshell 2.x contains Centos 7. End of life for Centos 7 is early 2024. Hence, Guestshell 4.x is a RockyLinux 9 based lxc container that will replace Guestshell 2.x. Guestshell 4.x is available in the following NX-OS releases as downloadable and default options.

| NXOS release      | Guestshell default package version | Guestshell downloadable option applicable |
|-------------------|------------------------------------|-------------------------------------------|
| 9.3(14) and above | 4.1 or above                       | Not needed                                |
| 10.2(6)           | 2.15                               | 4.0                                       |
| 10.2(7)           | 2.15                               | 4.1                                       |
| 10.2(8) and above | 4.1 or above                       | Not needed                                |

| NXOS release        | Guestshell default package version | Guestshell downloadable option applicable |
|---------------------|------------------------------------|-------------------------------------------|
| 10.3(4)             | 2.15                               | 4.0                                       |
| 10.3(5) and above   | 4.1 or above                       | Not needed                                |
| 10.4(1) and 10.4(2) | 2.15                               | 4.0                                       |
| 10.4(3) and above   | 4.1 or above                       | Not needed                                |
| 10.5(1) and above   | 4.1 or above                       | Not needed                                |



**Note** The rootfs size in Guestshell 4.x is 400 MB versus the 350 MB in Guestshell 2.x. Guestshell 4.x downloadable OVA is backward compactable to all releases running Guestshell 2.x as default.

### Upgrading from Guestshell 1.0 to Guestshell 2.x

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

### Downgrading NX-OS from Jacksonville release Guestshell 3.0

Beginning with Cisco NX-OS release 10.1(1), infrastructure version for Guestshell 3.0 support is increased to 1.11 (check with `show virtual-service` command). Therefore, Guestshell 3.0 OVA cannot be used in previous releases. If used, the **Install all** command will validate version mismatch and throws an error. It is recommended to destroy Guestshell 3.0 before downgrading to previous releases so that Guestshell 3.0 does not come up in previous releases.

### Upgrading from Guestshell 2.x to Guestshell 4.x Downloadable OVA

Guestshell 4.x can be downloaded from Cisco's official software download page and can be installed using command `guestshell upgrade` command.

Following table shows the guest shell releases:

**Table 3: Guest Shell Releases**

| Guest Shell Releases | NX-OS Supported Releases | Python Version(s) Supported |
|----------------------|--------------------------|-----------------------------|
| 2.x                  | 10.4.1                   | python2.7 and python 3.6    |
| 3.0                  | 10.1.x                   | python 3.6                  |
| 4.x                  | 10.4.1                   | python 3.9                  |

Use below commands to upgrade to Guestshell 4.x:

- Execute command `guestshell enable package <downloaded ova>` when guestshell is not installed.

- Execute command **guestshell upgrade package <downloaded ova>** when guestshell is installed and running.



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

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: Guestshell 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 Guestshell with the **guestshell destroy** command before reloading an older Cisco NX-OS image that does not support the Guestshell.

### Pre-Configured SSHD Service

The Guestshell 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 Guestshell 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 Guestshell from the NX-OS CLI might not work.

The following steps instantiate an OpenSSH server within the Guestshell 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 Guestshell sshd service for password-less access uses a randomized port starting at 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`.
3. Edit the `sshd-mgmt.service` file to match the following:
 

```
[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
```
4. 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
```
5. 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
```

6. (Optional) Check the configuration.

```
ss -tnldp | grep 2222
```

7. SSH into Guestshell:

```
ssh -p 2222 guestshell@10.122.84.34
```

8. Save the configuration across multiple Guestshell or switch reboots.

```
sudo systemctl enable sshd-mgmt.service
```

9. For passwordless SSH/SCP and remote execution, generate the public and private keys for the user ID you want to use 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
```

10. 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/*
```

11. 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 Guestshell 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 Guestshell 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, only network-admin users can access the Guest Shell by default. 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** The Guest Shell starting in 2.2(0.2) will dynamically create user accounts with the same 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 additional resources for the Guest Shell, the **guestshell resize** *{cpu | memory | rootfs}* command changes these limits.

| Resource | Default | Minimum/Maximum |
|----------|---------|-----------------|
| CPU      | 1%      | 1/620%          |
| Memory   | 400 MB  | 256/3840 MB     |
| Storage  | 200 MB  | 200/2000 MB     |

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 Guestshell

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

The Guestshell is populated with CentOS 7 Linux which provides the ability to yum install software packages built for this distribution. The Guestshell is pre-populated with many of the common tools that would naturally be expected on a networking device including **net-tools**, **iproute**, **tcpdump** and OpenSSH. For Guestshell 2.x, python 2.7.5 is included by default as is the PIP for installing additional python packages. In Guestshell 2.11, by default, python 3.6 is also included.

By default the Guestshell is a 64-bit execution space. If 32-bit support is needed, the `glibc.i686` package can be yum installed.

The Guestshell 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 as a convenience for running in a different namespace and getting information about which namespace/vrf a process is running in.

systemd is used to manage services in CentOS 8 environments, including the Guestshell.

## 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** For release 7.0(3)I5(2) using Starting with Guest Shell 2.2 (0.2), commands issued on the host through the **dohost** command are run with privileges based on the effective role of the Guest Shell user.
- Prior versions of Guest Shell will run command 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 a number of typical network utilities included by default and 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 may not be 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** 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/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 here.</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 is 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 nameserver and domain as appropriate for the network.

Example:

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

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

Example:

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

If the switch 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 will 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 2.7.5 (default, Jun 24 2015, 00:41:19)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] 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 versions up to 2.10 (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]# dnf install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
 Installing : scl-utils-20130529-19.el7.x86_64 1/1
 Verifying : scl-utils-20130529-19.el7.x86_64 1/1

Installed:
 scl-utils.x86_64 0:20130529-19.el7

Complete!

[root@guestshell admin]# dnf install -y centos-release-scl | tail
 Verifying : centos-release-scl-2-3.el7.centos.noarch 1/2
 Verifying : centos-release-scl-rh-2-3.el7.centos.noarch 2/2

Installed:
 centos-release-scl.noarch 0:2-3.el7.centos

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

Complete!

[root@guestshell admin]# dnf install -y rh-python36 | tail
warning: /var/cache/dnf/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 b1fb 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.e17
scl-utils-build.x86_64 0:20130529-19.e17
xml-common.noarch 0:0.6.3-39.e17
zip.x86_64 0:3.0-11.e17
```

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/bd/23c926cd341ee67d102a00aba99ae0f823e89c72b2190f27c11d4b7fb/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/c551b81dce15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/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/bc/a9/01ffebfb562e427466487d4bb1dbcc7ca55ec7510b22b4c51f14098443b8/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/d50ca96a08b00655a399c3ff1db9a7b75a24be7890bc9cf518e99/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/e8/74/6e4f91745020f967d09332b2b8c8b10090957334692ab88a4afe91b77f/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 dnf 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.

## Python 3 in Guest Shell versions up to 2.10 (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]# dnf install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
```

```

Installing : scl-utils-20130529-19.el7.x86_64 1/1
Verifying : scl-utils-20130529-19.el7.x86_64 1/1

Installed:
scl-utils.x86_64 0:20130529-19.el7

Complete!

[root@guestshell admin]# dnf install -y centos-release-scl | tail
Verifying : centos-release-scl-2-3.el7.centos.noarch 1/2
Verifying : centos-release-scl-rh-2-3.el7.centos.noarch 2/2

Installed:
centos-release-scl.noarch 0:2-3.el7.centos

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

Complete!

[root@guestshell admin]# dnf install -y rh-python36 | tail
warning: /var/cache/dnf/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 b1fb 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/23c926cc841ea657d0b2a00aba99ae0f828e89c72b2190f27c11d4b7fb/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/cd551d81d8e15200b6c4f41cd03869a46fe7226e7450af7a6545bfc474c9/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/a9/01ffebfb562e4274b6487b4bb1dbbc7ca55ac7510b22e4c51f1409844368/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/cf50cac98a0fb006c55a3993bffd9ba765a24b7890c9cf5fb9e99/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/e8/74/6e4f91745020f967d093321d289b10090957334692d88ea4afe91b77f/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 dnf 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/dnf/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.

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

For applications to be installed inside Guest Shell 3.0, go to the CentOS 8 repo at [http://mirror.centos.org/centos/8/BaseOS/x86\\_64/os/Packages/](http://mirror.centos.org/centos/8/BaseOS/x86_64/os/Packages/).

For applications to be installed inside Guest Shell 2.x, go to the CentOS 7 repo at [http://mirror.centos.org/centos/7/os/x86\\_64/Packages/](http://mirror.centos.org/centos/7/os/x86_64/Packages/).

For applications to be installed inside Guest Shell 4.x, go to the RockyLinux 9 repo at [https://mirrors.rockylinux.org/mirrorlist?arch=x86\\_64&repo=rocky-BaseOS-9.2](https://mirrors.rockylinux.org/mirrorlist?arch=x86_64&repo=rocky-BaseOS-9.2). Choose any one of mirror link and view the packages.

Dnf resolves the dependencies and installs all the required packages.

```
[guestshell@guestshell ~]$ sudo chvrf management dnf -y install glibc.i686
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: bay.uchicago.edu
* extras: pubmirrors.dal.corespace.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-sofotkn-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-sofotkn-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-sofotkn-freebl-3.16.2.3-9.el7.i686 2/2

Installed:
glibc.i686 0:2.17-78.el7

Dependency Installed:
nss-sofotkn-freebl.i686 0:3.16.2.3-9.el7

Complete!

```



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



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

## Security Posture for Virtual ServicesGuest Shell

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

Use of the Guest Shell in switches 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.

## Digitally Signed Application Packages

By default, Cisco network elements require applications to provide a valid Cisco digital signature at runtime. The Cisco digital signature ensures the integrity of Cisco-developed packages and applications. If a network-admin wants to use a software package (an OVA file) that was not Cisco-signed to create a virtual-service, he can configure the network element's image-signing policy under the virtual-service config submode using the **virtual-service** and **signing level unsigned** commands. It is recommended that an alternate means, such as SHA1 or MD5 checksums, be used to ensure the integrity of a software package (that is not Cisco-signed) before it is installed on the switch.



---

**Note** Cisco does not recommend using software packages that are not Cisco-signed.

---

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

The Cisco Nexus 30009000 Series switches support the configuration of a signing level policy to allow for unsigned OVA software packages. To allow unsigned and Cisco-signed packages for creating virtual-services, the network-admin can configure the following:

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



---

**Note** The Guest Shell software package has a Cisco signature and does not require this configuration.

---

## Kernel Vulnerability Patches

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



---

**Note** Cisco tracks the vulnerabilities for Guestshell 4.x (Rocky Linux 9) environment and will include future fixes when they are available from Rocky Linux.

---

## ASLR and X-Space Support

Cisco 30009000 NX-OS supports the use of Address Space Layout Randomization (ASLR) and Executable Space Protection (X-Space) for runtime defense. The software in Cisco-signed packages make use of this capability. If other software is installed on the system, it is recommended that it be built using a host OS and development toolchain that supports these technologies. Doing so reduces the potential attack surface that the software presents to potential intruders.

## Namespace Isolation

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

## Namespace Isolation

The Guest Shell environment runs within a Linux container that makes use of various namespaces to decouple the Guest Shell execution space from that of the host. Starting in the NX-OS 9.2(1) release, 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 gids 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 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 created by bob from the host is 65534. This indicates the actual id is in a range that is outside range of ids 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, it is recommend running applications with the least privilege needed to accomplish the assigned task. To help prevent unintended accesses, software added into the Guest Shell should follow this best practice.

All processes within a virtual servicethe 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 virtual servicethe Guest Shell follow:

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

- 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`

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

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 among all virtual services between Guest Shell and services on the host.

## Guest File System Access Restrictions

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

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.

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

`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 4: Guest Shell CLI Commands*

Commands	Description

Commands	Description
<b>guestshell enable</b> { <b>package</b> [ <i>guest shell OVA file</i>   <i>rootfs-file-URI</i> ]}	<ul style="list-style-type: none"> <li>• 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 <b>guestshell disable</b> command.</li> <li>• When <i>rootfs-file-URI</i> is specified:            Imports a Guest Shell <b>rootfs</b> when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.</li> </ul>
<b>guestshell export rootfs package</b> <i>destination-file-URI</i>	Exports a Guest Shell <b>rootfs</b> file to a local URI (bootflash, USB1, etc.). (7.0(3)I7(1) and later releases)
<b>guestshell disable</b>	Shuts down and disables the Guest Shell.

Commands	Description
<p><b>guestshell upgrade</b> {<b>package</b> [<i>guest shell OVA file</i>   <i>rootfs-file-URI</i>]}</p>	<ul style="list-style-type: none"> <li>• 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 <b>guestshell destroy</b> command followed by a <b>guestshell enable</b> 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 prior to carrying out the upgrade command.</p> </li> <li>• When <i>rootfs-file-URI</i> is specified: <p>Imports a Guest Shell <b>rootfs</b> file when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.</p> </li> </ul>
<p><b>guestshell reboot</b></p>	<p>Deactivates the Guest Shell and then reactivates it.</p> <p>You are prompted for a confirmation prior to carrying out the reboot command.</p> <p><b>Note</b> This is the equivalent of a <b>guestshell disable</b> command followed by a <b>guestshell enable</b> command in exec mode.</p> <p>This is useful when processes inside the Guest Shell have been stopped and need to be restarted. The <b>run guestshell</b> 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
<b>guestshell destroy</b>	<p>Deactivates and uninstalls the Guest Shell. All resources associated with the Guest Shell are returned to the system. The <b>show virtual-service global</b> command indicates when these resources become available.</p> <p>Issuing this command results in a prompt for a confirmation prior to carrying out the destroy command.</p>
<b>guestshell</b> <b>run guestshell</b>	Connects to the Guest Shell that is already running with a shell prompt. No username/password is required.
<b>guestshell run</b> <i>command</i> <b>run guestshell</b> <i>command</i>	<p>Executes a Linux/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>
<b>guestshell resize</b> [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><b>Note</b> Resize values are cleared when the <b>guestshell destroy</b> command is used.</p>
<b>guestshell sync</b>	On systems that have active and standby supervisors, this command synchronizes the Guest Shell contents from the active supervisor to the standby supervisor. The network-admin issues this command when the Guest Shell rootfs has been set up to a point that they would want the same rootfs used on the standby supervisor when it becomes the active supervisor. If this command is not used, the Guest Shell is freshly installed when the standby supervisor transitions to an active role using the Guest Shell package available on that supervisor.
<b>virtual-service reset force</b>	<p>In the event that the guestshell or virtual-services cannot be managed, even after a system reload, the reset command is used to force the removal of the Guest Shell and all virtual-services. The system needs to be reloaded for the cleanup to happen. No Guest Shell or additional virtual-services can be installed or enabled after issuing this command until after the system has been reloaded.</p> <p>You are prompted for a confirmation prior to 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+`.

---




---

**Note** Virtual-service commands that do not pertain to the Guest Shell are being deprecated. These commands have been hidden in the NX-OS 9.2(1) release and will be removed in future releases.

The following exec keywords are being deprecated:

```
virtual-service ?
connect Request a virtual service shell
install Add a virtual service to install database
uninstall Remove a virtual service from the install database
upgrade Upgrade a virtual service package to a different version
```

```
show virtual-service ?
detail Detailed information config)
```

The following config keywords are being deprecated:

```
(config) virtual-service ?
WORD Virtual service name (Max Size 20)
```

```
(config-virt-serv)# ?
activate Activate configured virtual service
description Virtual service description
```

---

## 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 the Guest Shell is destroyed and the system is reloaded, the Guest Shell remains destroyed.

```

switch# show virtual-service list
Virtual Service List:
Name Status Package Name

guestshell+ Deactivated guestshell.ova

switch# guestshell destroy

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

switch# show virtual-service list
Virtual Service List:

```




---

**Note** The Guest Shell can be re-enabled with the **guestshell enable** command.

---




---

**Note** 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 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

Beginning in the NX-OS 9.2(1) release, 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. Once you have done this, the Guest Shell and virtual-service commands will be 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.

### Enabling the Guest Shell on Cisco Nexus 3000 with Compacted Image

The Guest Shell software is not available in a Cisco NX-OS image that has been compacted for the Cisco Nexus 3000 Series switches with 1.6 GB bootflash and 4 GB RAM. You can still use the Guest Shell in this case, but you will need to download the software package from [software.cisco.com](http://software.cisco.com) for the Cisco NX-OS release, then you will need to copy it onto the Cisco Nexus 3000 Series switch and enable it.

For more information on compacted images, refer to the *Cisco Nexus 3000 Series NX-OS Software Upgrade and Downgrade Guide, Release 9.2(1)*.

The Guest Shell software installs onto the bootflash of the switch. To create as much free bootflash space as possible, put the downloaded `guestshell.ova` file onto the `volatile:` storage media. Once the Guest

Shell is successfully activated, the `guestshell.ova` file can be deleted. It will not be needed again unless the Guest Shell is destroyed at some point and needs to be re-installed.

For example:

```
switch# copy scp://admin@1.2.3.4/guestshell.ova volatile: vrf management
guestshell.ova 100% 55MB 10.9MB/s 00:05
Copy complete, now saving to disk (please wait)...
Copy complete.

switch# dir volatile: | inc .ova
57251840 Jun 22 11:56:51 2018 guestshell.ova

switch# guestshell enable package volatile:guestshell.ova
2018 Jun 7 19:13:03 n3x-164 %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual service
'guestshell+'
2018 Jun 7 19:13:56 n3x-164 %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2018 Jun 7 19:13:56 n3x-164 %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
'guestshell+'
2018 Jun 7 19:15:34 n3x-164 %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'

switch# del volatile:guestshell.ova
Do you want to delete "/guestshell.ova" ? (yes/no/abort) [y] y

switch# guestshell
[admin@guestshell ~]$
```

## Replicating the Guest Shell

Beginning with Cisco NX-OS release 7.0(3)I7(1), a Guest Shell **rootfs** that is customized on one switch 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, etc.).

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 stored on local storage (bootflash, USB, etc.).

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 additional parameters) when the Guest Shell has already been installed.
  - Use the **guestshell enable** command (without additional 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. It is embedded into the Guest Shell **rootfs** in the /cisco directory. It is not a complete descriptor for the Guest Shell container. It only contains some of the parameters that are user modifiable.

Example of a Guest Shell import YAML file:

```

import-schema-version: "1.0"
info:
 name: "GuestShell"
 version: "2.2(0.3)"
 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 required for hosting the Guest Shell. The value "/" for the target-dir in the example identifies the disk as the **rootfs**.



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

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 [Guest Shell 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 : rootfs_puppet
Path : usb2:/rootfs_puppet
Application
Name : GuestShell
Installed version : 3.0(0.0)
Description : Exported GuestShell: 20170613T173648Z
Signing
Key type : Unsigned
Method : Unknown

```

```
Licensing
 Name : None
 Version : None
```

## Verifying Virtual Service and Guest Shell Information

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

Command	Description
<pre><b>show virtual-service global</b>  switch# <b>show virtual-service global</b>  Virtual Service Global State and Virtualization Limits:  Infrastructure version : 1.11 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 (%) 20 1 19 memory (MB) 3840 256 3584 bootflash (MB) 8192 200 7992 switch#</pre>	Displays the global state and limits for virtual services.
<pre><b>show virtual-service list</b>  switch# <b>show virtual-service list *</b>  Virtual Service List:  Name                Status      Package Name ----- guestshell+         Activated   guestshell.ova chef                 Installed   chef-0.8.1-n9000-spa-k9.ova</pre>	Displays a summary of the virtual services, the status of the virtual services, and installed software packages.

Command	Description
<pre> <b>show guestshell detail</b>  switch# <b>show guestshell detail</b> Virtual service guestshell+ detail   State           : Activated   Package information     Name           : guestshell.ova     Path           : /isan/bin/guestshell.ova   Application     Name           : GuestShell     Installed version : 3.0(0.0)     Description    : Cisco Systems Guest Shell   Signing     Key type      : Cisco key     Method       : SHA-1   Licensing     Name         : None     Version      : None   Resource reservation     Disk        : 400 MB     Memory      : 256 MB     CPU         : 1% system CPU    Attached devices   Type          Name          Alias   -----   Disk          _rootfs   Disk          /cisco/core   Serial/shell   Serial/aux   Serial/Syslog          serial2   Serial/Trace          serial3 </pre>	<p>Displays details about the guestshell 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 Get Into Guest Shell After Downgrade to 7.0(3)I7

If you downgrade from the NX-OS 9.2(1) release to the NX-OS 7.0(3)7 release image (which does not have user namespace support) while the Guest Shell is in the process of activating or deactivating, you may run into the following condition where the Guest Shell activates, but you are unable to get into the Guest Shell. The reason for this issue is that if a reload is issued while the Guest Shell is in transition, the files within the Guest Shell can't get shifted back into an id range that is usable for NX-OS releases that don't have user namespace support.

```

switch# guestshell
Failed to mkdir .ssh for admin

```

```

admin RSA add failed
ERROR: Failed to connect with Virtual-service 'guestshell+'
switch#
switch# sh virt list

Virtual Service List:
Name Status Package Name

guestshell+ Activated guestshell.ova

switch# run bash ls -al /isan/vdc_1/virtual-instance/guestshell+/rootfs/
drwxr-xr-x 24 11000 11000 1024 Apr 11 10:44 .
drwxrwxrwx 4 root root 80 Apr 27 20:08 ..
-rw-r--r-- 1 11000 11000 0 Mar 21 16:24 .autorelabel
lrwxrwxrwx 1 11000 11000 7 Mar 21 16:24 bin -> usr/bin

```

To recover from this issue without losing the contents of the Guest Shell, reload the system with the previously-running NX-OS 9.2(x) image and let the Guest Shell get to the `Activated` state before reloading the system with the NX-OS 7.0(3)I7 image. Another option is to disable the Guest Shell while running NX-OS 9.2(x) and re-enable it after reloading with 7.0(3)I7.

If you do not have anything to preserve in the Guest Shell and you just want to recover it, you can destroy and recreate it without needing to change images.

### Unable to Access Files on bootflash from root in the Guest Shell

You may find that you are unable to access files on 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 may be due to the fact that, because the user namespace is being used to protect the host system, 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

# Python API

---

- [Information About the Python API, on page 63](#)
- [Using Python, on page 63](#)

## Information About the Python API

Beginning with Cisco NX-OS Release 9.3(5), Python 3 is now supported. Python 2.7 will continue to be supported. We recommend that you use the **python3** command for new scripts.

The Cisco Nexus 3500 platform switches support Python v2.7.11 and v3.7.3 in both interactive and noninteractive (script) modes and are available in the Guest Shell.

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 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 can be accessed from the Bash shell.

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

For information about using Python with Cisco Nexus devices, see the Cisco Nexus 9000 Series Python SDK User Guide and API Reference at this URL: <https://developer.cisco.com/site/nx-os/docs/apis/python/>. This information applies also to the Cisco Nexus 3000 Series devices.

## 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-1.0.0-9.2.1.lib32_x86.rpm`) from [https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86\\_64/](https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86_64/). Refer to the "Manage Feature RPMs" section for instructions on 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
```

The following is an example of how to display information about the Cisco Python Package for Python 3:

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
 cisco
```

```

PACKAGE CONTENTS
acl
bgp
buffer_depth_monitor
check_port_discards
cisco_secret
feature
historys
interface
ipaddress
key
line_parser
mac_address_table
md5sum
nxcli
nxos_cli
ospf
routemap
routes
section_parser
ssh
system
tacacs
transfer
vlan
vrf

CLASSES
builtins.dict(builtins.object)
cisco.history.History
builtins.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 5: CLI Command APIs**

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

API	Description
<b>clid()</b> Example: <pre>json_string = clid ("cli-command")</pre>	Returns JSON output for <b>cli-command</b> , if XML support exists for the command, otherwise an exception is thrown.  <b>Note</b> This API can be useful when searching the output of show commands.
<b>clip()</b> Example: <pre>clip ("cli-command")</pre>	Prints the output of the CLI command directly to stdout and returns nothing to Python.  <b>Note</b> <pre>clip ("cli-command")</pre> is equivalent to <pre>r=cli("cli-command") print r</pre>

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

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

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

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

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

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




---

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

---

## Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python 2 from the CLI:




---

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

---



**Important** Python 2.7 is End of Support, future Cisco NX-OS software deprecates Python 2.7 support. We recommend for new scripts to use **python3** instead. Type **python3** to use the new shell.

```
switch# python
switch# python
```

Warning: Python 2.7 is End of Support, and future NXOS software will deprecate python 2.7 support. It is recommended for new scripts to use 'python3' instead. Type "python3" to use the new shell.

```
Python 2.7.11 (default, Jun 4 2020, 09:48:24)
[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 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
... intf=intflist['TABLE_interface']['ROW_interface'][i]
... i=i+1
... if intf['state'] == 'up':
... print intf['interface']
...
mgmt0
loopback1
>>>
```

The following example shows how to invoke Python 3 from the CLI:

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
... intf=intflist['TABLE_interface']['ROW_interface'][i]
... i=i+1
... if intf['state'] == 'up':
... print(intf['interface'])
...
mgmt0
loopback1
>>>
```

## 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:**

```
>>> r = cli('where detail')
>>> print(r)
mode:
username: admin
vdc: switch
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]))
...
header_str - Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2020, 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.
bios_ver_str - 07.67
kickstart_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
nxos_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
bios_cmpl_time - 01/29/2020
kick_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
nxos_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
```

```

kick_cmpl_time - 5/10/2020 21:00:00
nxos_cmpl_time - 5/10/2020 21:00:00
kick_tmstamp - 05/12/2020 07:08:44
nxos_tmstamp - 05/12/2020 07:08:44
chassis_id - Nexus9000 93180YC-EX chassis
cpu_name - Intel(R) Xeon(R) CPU @ 1.80GHz
memory - 24632252
mem_type - kB
proc_board_id - FDO22280FFK
host_name - switch
bootflash_size - 53298520
kern_uptm_days - 0
kern_uptm_hrs - 0
kern_uptm_mins - 19
kern_uptm_secs - 34
rr_usecs - 641967
rr_ctime - Tue May 12 09:52:28 2020
rr_reason - Reset Requested by CLI command reload
rr_sys_ver - 9.4(1)
rr_service - None
plugins - Core Plugin, Ethernet Plugin
manufacturer - Cisco Systems, Inc.
>>>

```

## 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 3500 platform switches also support the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

This example shows the script first and then executing it. Saving is like bringing any file to the bootflash.

```

switch# show file bootflash:scripts/deltaCounters.py
#!/isan/bin/python3
from cli import *
import sys, time
ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'
out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print ('row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast')
print ('=====')
print (' %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc))
print ('=====')
i = 0
while (i < count):
 time.sleep(delay)
 out = json.loads(clid(cmd))
 rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
 rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
 rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])

```

```

txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
i += 1
print ('%-3d %8d %8d %8d %8d %8d %8d' % (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew -
rxbc, txucNew - txuc, txmcNew - txmc, txbcNew - txbc))

switch# python bootflash:scripts/deltaCounters.py mgmt0 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
=====
 291 8233 1767 185 57 2
=====
1 1 4 1 1 0 0
2 2 5 1 2 0 0
3 3 9 1 3 0 0
4 4 12 1 4 0 0
5 5 17 1 5 0 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 3500 platform switches, 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: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

version 9.3(5) Bios:version 07.67
event manager applet a1
 event cli match "show clock"
 action 1 cli python bootflash:pydate.py

switch# show file logflash:vdc_1/event_archive_1 | last 33

eem_event_time:06/25/2020,15:34:24 event_type:cli event_id:24 slot:active(1) vdc
:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1

```



```

stty: standard input: Inappropriate ioctl for device
Executing the following commands succeeded:
 python bootflash:pydate.py
Completed executing policy al
Event Id:24 event type:10241 handling completed

```

- 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:al
eem_param_info:command = "exshow clock"
Starting with policy al
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 3500 platform 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

Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.

Python 2.7.11 (default, Jun 4 2020, 09:48:24)
[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

- 

### Example of Running Script with Scheduler

-



## CHAPTER 6

# Python API

---

- [About the Python API](#) , on page 73
- [Using Python](#), on page 73

## 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 30009000 Series devices support Python v2.7.5 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.

For information about using Python with Cisco Nexus devices, see the Cisco Nexus 9000 Series Python SDK User Guide and API Reference at this URL: <https://developer.cisco.com/site/nx-os/docs/apis/python/>. This information applies also to the Cisco Nexus 3000 Series devices.

## 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-1.0.0-9.2.1.lib32_x86.rpm`) from [https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86\\_64/](https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86_64/). Refer to the "Manage Feature RPMs" section for instructions on 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
```

The following is an example of how to display information about the Cisco Python Package for Python 3:

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
 cisco
```

```

PACKAGE CONTENTS
acl
bgp
buffer_depth_monitor
check_port_discards
cisco_secret
feature
historys
interface
ipaddress
key
line_parser
mac_address_table
md5sum
nxcli
nxos_cli
ospf
routemap
routes
section_parser
ssh
system
tacacs
transfer
vlan
vrf

CLASSES
builtins.dict(builtins.object)
cisco.history.History
builtins.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 6: CLI Command APIs**

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

API	Description
<b>clid()</b> 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.  <b>Note</b> This API can be useful when searching the output of show commands.
<b>clip()</b> Example: <pre>clip ("cli-command")</pre>	Prints the output of the CLI command directly to stdout and returns nothing to Python.  <b>Note</b> <pre>clip ("cli-command")</pre> is equivalent to <pre>r=cli("cli-command") print r</pre>

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

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

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

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

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

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




---

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

---

## Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python 2 from the CLI:




---

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

---



**Important** Python 2.7 is End of Support, Future NX-OS software deprecates Python 2.7 support. We recommend for new scripts to use **python3** instead. Type **python3** to use the new shell.

The following example shows how to invoke Python 3 from the CLI:

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
... intf=intflist['TABLE_interface']['ROW_interface'][i]
... i=i+1
... if intf['state'] == 'up':
... print(intf['interface'])
...
mgmt0
loopback1
>>>
```

## 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:

```
>>> r = cli('where detail')
>>> print(r)
mode:
username: admin
vdc: switch
```

```
routing-context vrf: default
```

```
>>>
```

#### Example 4:

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
... print("%30s - %s" % (k,out[k]))
...
header_str - Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2020, 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.
bios_ver_str - 07.67
kickstart_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
nxos_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
bios_cmpl_time - 01/29/2020
kick_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
nxos_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
kick_cmpl_time - 5/10/2020 21:00:00
nxos_cmpl_time - 5/10/2020 21:00:00
kick_tmstamp - 05/12/2020 07:08:44
nxos_tmstamp - 05/12/2020 07:08:44
chassis_id - Nexus9000 93180YC-EX chassis
cpu_name - Intel(R) Xeon(R) CPU @ 1.80GHz
memory - 24632252
mem_type - kB
proc_board_id - FDO22280FFK
host_name - switch
bootflash_size - 53298520
kern_uptm_days - 0
kern_uptm_hrs - 0
kern_uptm_mins - 19
kern_uptm_secs - 34
rr_usecs - 641967
rr_ctime - Tue May 12 09:52:28 2020
rr_reason - Reset Requested by CLI command reload
rr_sys_ver - 9.4(1)
rr_service - None
plugins - Core Plugin, Ethernet Plugin
manufacturer - Cisco Systems, Inc.
>>>
```



## 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 switch also supports the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

This example shows the script first and then executing it. Saving is like bringing any file to the bootflash.

```
switch# show file bootflash:scripts/deltaCounters.py
#!/isan/bin/python3
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:scripts/deltaCounters.py mgmt0 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
=====
 291 8233 1767 185 57 2
=====
1 1 4 1 1 0 0
2 2 5 1 2 0 0
3 3 9 1 3 0 0
4 4 12 1 4 0 0
5 5 17 1 5 0 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 switches, 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: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

version 9.3(5) Bios:version 07.67
event manager applet a1
 event cli match "show clock"
 action 1 cli python bootflash:pydate.py

switch# show file logflash:vdc_1/event_archive_1 | last 33

eem_event_time:06/25/2020,15:34:24 event_type:cli event_id:24 slot:active(1) vdc
:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
stty: standard input: Inappropriate ioctl for device
Executing the following commands succeeded:
 python bootflash:pydate.py
Completed executing policy a1
Event Id:24 event type:10241 handling completed

```

- 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 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

Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.

Python 2.7.11 (default, Jun 4 2020, 09:48:24)
[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:

Python 3 example.

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
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')
17
>>> 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# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','w')
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
PermissionError: [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')
'Warning: No NTP peer/server configured. Time may be out of sync.\n15:39:39.513 UTC Thu Jun
 25 2020\nTime source is NTP\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Running configuration last done at: Thu Jun 25 15:39:49 2020
!Time: Thu Jun 25 15:39:55 2020

version 9.3(5) Bios:version 07.67

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
```

Python 3 example.

```
#!/bin/env python3
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/test.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 Sat Jun 13 04:29:38 2020
switch# 2020 Jun 13 04:29:41 switch %USER-1-SYSTEM_MSG: No user ran /bootflash/scripts/test.py
 on : switch - nxpython
switch# show scheduler schedule
Schedule Name : testplan

User Name : admin
Schedule Type : Run every 0 Days 0 Hrs 4 Mins
Start Time : Sat Jun 13 04:29:38 2020
Last Execution Time : Sat Jun 13 04:29:38 2020
Last Completion Time: Sat Jun 13 04:29:41 2020
Execution count : 1

Job Name Last Execution Status

testplan Success (0)
=====
switch#

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

---

- [About Tcl, on page 85](#)
- [Running the Tclsh Command, on page 87](#)
- [Navigating Cisco NX-OS Modes from the Tclsh Command, on page 88](#)
- [Tcl References, on page 90](#)

## 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 switches.

## Tclsh Command Help

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

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

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

switch-tcl#
```



---

**Note** In the 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.

---

### SUMMARY STEPS

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

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>tclsh</b> [ <b>bootflash:filename</b> [ <i>argument ...</i> ]]  <b>Example:</b> <pre>switch# tclsh ? &lt;CR&gt; bootflash: The file to run</pre>	Starts a Tcl shell.  If you run the <b>tclsh</b> 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 <b>tclquit</b> or <b>Ctrl-C</b> .  If you run the <b>tclsh</b> command with arguments, the first argument is the name of a script file containing Tcl commands and any additional arguments are made available to the script as variables.

**Example**

The following example shows an interactive Tcl shell:

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

switch-tcl# exit
switch#
```

The following example shows how to run a Tcl script:

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

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

switch#
```

## Navigating Cisco NX-OS Modes from the Tclsh Command

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

## SUMMARY STEPS

1. **tclsh**
2. **configure terminal**
3. **tclquit**

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>tclsh</b> <b>Example:</b> <pre>switch# tclsh switch-tcl#</pre>	Starts an interactive Tcl shell.
<b>Step 2</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-tcl# configure terminal switch(config-tcl)#</pre>	Runs a Cisco NX-OS command in the Tcl shell, changing modes.  <b>Note</b> The Tcl prompt changes to indicate the Cisco NX-OS command mode.
<b>Step 3</b>	<b>tclquit</b> <b>Example:</b> <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

# Ansible

- [Prerequisites](#), on page 91
- [About Ansible](#), on page 91
- [Cisco Ansible Module](#), on page 91

## Prerequisites

Go to [https://docs.ansible.com/ansible/latest/getting\\_started/index.html](https://docs.ansible.com/ansible/latest/getting_started/index.html) for installation requirements for supported control environments.

## About Ansible

Ansible is an open-source IT automation engine that automates cloud provisioning, configuration management, application deployment, intraservice orchestration, and other IT needs.

Ansible uses small programs that are called Ansible modules to make API calls to your nodes, and apply configurations that are defined in playbooks.

By default, Ansible represents what machines it manages using a simple INI file that puts all your managed machines in groups of your own choosing.

More information can be found from Ansible:

Ansible	<a href="https://www.ansible.com/">https://www.ansible.com/</a>
Ansible Automation Solutions. Includes installation instructions, playbook instructions and examples, module lists, and so on.	<a href="https://docs.ansible.com/">https://docs.ansible.com/</a>

## Cisco Ansible Module

There are multiple Cisco NX-OS-supported modules and playbooks for Ansible, as per the following table of links:

NX-OS developer landing page.	<a href="#">Configuration Management Tools</a>
-------------------------------	------------------------------------------------

Ansible NX-OS playbook examples	<a href="#">Repo for ansible nxos playbooks</a>
Ansible NX-OS network modules	<a href="#">nxos network modules</a>



## CHAPTER 9

# Puppet Agent

This chapter includes the following sections:

- [About Puppet, on page 93](#)
- [Prerequisites, on page 94](#)
- [Puppet Agent NX-OS Environment, on page 94](#)
- [ciscopuppet Module, on page 94](#)

## About Puppet

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

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

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

More information can be found from Puppet Labs:

Puppet Labs	<a href="https://puppetlabs.com">https://puppetlabs.com</a>
Puppet Labs FAQ	<a href="https://puppet.com/blog/how-get-started-puppet-enterprise-faq/">https://puppet.com/blog/how-get-started-puppet-enterprise-faq/</a> <a href="https://puppet.com/products/faq">https://puppet.com/products/faq</a>
Puppet Labs Documentation	<a href="https://puppet.com/docs">https://puppet.com/docs</a>

## Prerequisites

The following are prerequisites for the Puppet Agent:

- You must have a switch and operating system software release that supports the installation.
  - Cisco Nexus 3600 platform switches.
  - Cisco Nexus 3500 platform switches
  - Cisco Nexus 3100 platform switches.
  - Cisco Nexus 3000 Series switches.
  - Cisco NX-OS Release 7.0(3)I2(1) or later.
- You must have the required disk storage available on the device for virtual services installation and deployment of Puppet Agent.
  - A minimum of 450MB free disk space on bootflash.
- You must have Puppet Primary server with Puppet 4.0 or later.
- You must have Puppet Agent 4.0 or later.

## Puppet Agent NX-OS Environment

The Puppet Agent software must be installed on a switch in the Guest Shell (the Linux container environment running CentOS). The Guest Shell provides a secure, open execution environment that is decoupled from the host.

Starting with the Cisco NX-OS Release 9.2(1), the Bash-shell (native WindRiver Linux environment underlying Cisco NX-OS) install of Puppet Agent is no longer supported.

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

Puppet Agent: Installation & Setup on Cisco Nexus switches (Manual Setup)	<a href="https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md">https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md</a>
---------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## ciscopuppet Module

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

The ciscopuppet module is available on Puppet Forge.

The following provide additional information about the ciscopuppet module installation procedures:



ciscopuppet Module location (Puppet Forge)	<a href="#">Puppet Forge</a>
Resource Type Catalog	<a href="#">Cisco Puppet Resource Reference</a>
ciscopuppet Module: Source Code Repository	<a href="#">Cisco Network Puppet Module</a>
ciscopuppet Module: Setup & Usage	<a href="#">Cisco Puppet Module::README.md</a>
Puppet Labs: Installing Modules	<a href="https://puppet.com/docs/puppet/7/modules_installing.html">https://puppet.com/docs/puppet/7/modules_installing.html</a> <a href="https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html">https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html</a>
Puppet NX-OS Manifest Examples	<a href="#">Cisco Network Puppet Module Examples</a>
NX-OS developer landing page.	<a href="#">Configuration Management Tools</a>





## CHAPTER 10

# SaltStack

---

This chapter contains the following topics:

- [About SaltStack, on page 97](#)
- [Guidelines and Limitations, on page 98](#)
- [Cisco NX-OS Environment for SaltStack, on page 98](#)
- [Enabling NX-API for SaltStack, on page 99](#)
- [Installing SaltStack for NX-OS, on page 99](#)

## About SaltStack

The Cisco Nexus switches support SaltStack through NX-OS. For information about Cisco NX-OS releases that support SaltStack, see <https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#step-1-verify-platform-and-software-version-support>.

SaltStack is a free and open source automation framework for configuration, management, and remote execution of servers and other network devices. The SaltStack framework consists of a server that is called the Salt primary, and Salt nodes that run client programs, called minions. The Cisco Nexus switch (switch) is a Salt node, not the Salt primary.

SaltStack minions can run either on-box or off-box, respective to the switch, to execute the configuration or management operations:

- On-box, the minions run in the switch's Bash shell. These native minions receive and execute remote commands from the primary, and relay the command's results to the primary. In an on-box deployment, the minions are enabled in the switch's Guest shell.
- Off-box, a different type of minion, a proxy minion, runs over an SSH connection to the switch or through the NX-API. The proxy minion, either the SSH proxy minion or the NX-API proxy minion, receives and executes the commands. The proxy then relays the command's results to the primary.

Keys are used to ensure security between the Salt primary and the minions running on the Cisco Nexus switch. When the Salt primary initiates its connection with a minion running on the Cisco Nexus switch, it first passes a key. The minion receives the key, then computes the correct response, and transmits the key back to the primary. The primary also has computed the correct response value for the key. When the primary receives the key from the minion, if the keys match, the session is open. The Salt primary can then send commands. Sessions are not persistent across power cycles or reboots.

SaltStack manages and configures the switch through execution modules and salt states, which affect the switch's CLI, properties, and features. For example, through the modules, SaltStack can be used to upgrade

the Cisco Nexus switches. The Salt primary sends commands programmatically to leverage automation and scalability.

For more information, consult the following documentation:

SaltStack	<a href="https://www.saltstack.com/">https://www.saltstack.com/</a>
SaltStack Documentation	<a href="https://docs.saltstack.com/en/latest/">https://docs.saltstack.com/en/latest/</a>
Cisco Nexus Salt Minion Installation and Configuration Guide	<a href="https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst">https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst</a>

## About NX-OS and SaltStack

Salt Open is the open source, community edition of the Salt configuration management and distributed remote execution system. Cisco NX-OS provides an intermediate layer between the physical switch and the Salt Open software. Cisco NX-OS and Salt Open interoperate to provide the API and command-execution layer between Salt minions and Cisco Nexus switches. Cisco NX-OS hosts the minions and enables them to run as follows:

- On the switch, the Cisco NX-OS guest shell hosts SaltStack minions and provides automated orchestration of one or more switches through a unified interface. The minion running in the guest shell is a native minion and it connects over the NX-API the UNIX Domain Socket (UDS).
- Off the switch, the Salt primary runs the Salt Open software on a network device and communicates with NX-OS through SSH (the SSH proxy minion) or NX-API over HTTPS (the NX-API proxy minion). Cisco NX-OS interprets the commands, performs required configuration tasks, and reports success or failure back to the appropriate proxy minion. The proxy minion, in turn, transmits this data back to the Salt primary.

## Guidelines and Limitations

The following are the guidelines and limitations for implementing SaltStack on the Cisco Nexus switches:

- If you are running SaltStack over SSH or NX-API HTTPS, enable the NX-API feature (**feature nxapi**) before you run Salt.
- The Salt primary listens for minions on port 4506. Make sure that this port is open (unblocked) and not used by another service.

## Cisco NX-OS Environment for SaltStack

The Cisco NX-OS environment is different depending on whether you are running Salt on box or off box.

- For on-box management of the switch, you must install the SaltStack minion RPM in the Guest Shell, which is the hosting environment for the minion.
- For off-box management of the switch, SSH or NX-API must be enabled in NX-OS.

For more information, such as which Cisco Nexus switches support SaltStack, go to <https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#step-1-verify-platform-and-software-version-support>.

# Enabling NX-API for SaltStack

## Before you begin

For proxy minions running over SSH or NX-API HTTPS, the NX-API feature must be enabled for SaltStack to function. By default, NX-API is enabled. The following instructions are provided in case you need to reenble it.

## SUMMARY STEPS

1. `config terminal`
2. `feature nxapi`

## DETAILED STEPS

### Procedure

	Command or Action	Purpose
Step 1	<b>config terminal</b> <b>Example:</b> <pre>switch-1# config terminal</pre> Enter configuration commands, one per line. End with CNTL/Z. <pre>switch-1(config)#</pre>	Enters configuration mode.
Step 2	<b>feature nxapi</b> <b>Example:</b> <pre>switch-1# feature nxapi</pre> <pre>switch-1#(config)#</pre>	Enables NX-API for proxy minions.

### What to do next

Install SaltStack.

# Installing SaltStack for NX-OS

Use the following installation guide to install and bring up SaltStack on the Cisco Nexus switches:

<https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#cisco-nexus-salt-minion-installation-and-configuration-guide>





# CHAPTER 11

## Using Chef Client with Cisco NX-OS

This chapter includes the following sections:

- [About Chef, on page 101](#)
- [Prerequisites, on page 101](#)
- [Chef Client NX-OS Environment, on page 102](#)
- [cisco-cookbook, on page 102](#)

### About Chef

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

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

The following references provide more information from Chef:

Topic	Link
Chef home	<a href="https://www.chef.io">https://www.chef.io</a>
Chef overview	<a href="https://docs.chef.io/chef_overview.html">https://docs.chef.io/chef_overview.html</a>
Chef documentation (all)	<a href="https://docs.chef.io/">https://docs.chef.io/</a>

### Prerequisites

The following are prerequisites for Chef:

- You must have a Cisco switch and operating system software release that supports the installation:
  - Cisco Nexus 3500 platform switch
  - Cisco NX-OS Release 6.1(2)I3(4) or higher
- You must have the required disk storage available on the device for Chef deployment:
  - A minimum of 500 MB free disk space on bootflash
- You need a Chef server with Chef 12.4.1 or higher.
- You need Chef Client 12.4.1 or higher.

## Chef Client NX-OS Environment

The chef-client software must be installed on Cisco Nexus switches. Customers can install chef-client in one of the Linux environments provided by the Cisco Nexus switch:

- Bash Shell — This is the native WindRiver Linux environment underlying Cisco NX-OS.
- Guest Shell — This is a secure Linux container environment running CentOS. Its advantage is a secure, open execution environment that is decoupled from the host.

The workflow for both use cases is similar.

The following documents provide step-by-step guidance on agent software download, installation, and setup:

Topic	Link
Chef Client (Native)	Latest information on Client RPM is available <a href="#">here</a> .
Chef Client (Guest Shell, CentOs7)	Latest information on Client RPM is available <a href="#">here</a> .
Chef Client: Installation and setup on Cisco Nexus platform (manual setup)	<a href="#">cisco-cookbook::README-install-agent.md</a>
Chef Client: Installation and setup on Cisco Nexus platform (automated installation using the Chef provisioner)	<a href="#">cisco-cookbook::README-chef-provisioning.md</a>

## cisco-cookbook

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

cisco-cookbook can be found on Chef Supermarket.

The following documents provide additional detail for cisco-cookbook and generic cookbook installation procedures:



Topic	Link
cisco-cookbook location	<a href="https://supermarket.chef.io/cookbooks/cisco-cookbook">https://supermarket.chef.io/cookbooks/cisco-cookbook</a>
Resource Type Catalog	<a href="https://github.com/cisco/cisco-network-chef-cookbook#resource-by-tech">https://github.com/cisco/cisco-network-chef-cookbook#resource-by-tech</a>
cisco-cookbook: Source Code Repository	<a href="https://github.com/cisco/cisco-network-chef-cookbook">https://github.com/cisco/cisco-network-chef-cookbook</a>
cisco-cookbook: Setup and usage	<a href="#">cisco-cookbook::README.md</a>
Chef Supermarket	<a href="https://supermarket.chef.io">https://supermarket.chef.io</a>
NX-OS developer landing page.	<a href="#">Configuration Management Tools</a>





## CHAPTER 12

# Using Docker with Cisco NX-OS

---

This chapter contains the following topics:

- [About Docker with Cisco NX-OS, on page 105](#)
- [Guidelines and Limitations, on page 105](#)
- [Prerequisites for Setting Up Docker Containers Within Cisco NX-OS, on page 106](#)
- [Starting the Docker Daemon, on page 106](#)
- [Configure Docker to Start Automatically, on page 107](#)
- [Starting Docker Containers: Host Networking Model, on page 108](#)
- [Starting Docker Containers: Bridged Networking Model, on page 109](#)
- [Mounting the bootflash and volatile Partitions in the Docker Container, on page 110](#)
- [Enabling Docker Daemon Persistence on Enhanced ISSU Switchover, on page 110](#)
- [Enabling Docker Daemon Persistence on the Cisco Nexus Platform Switches Switchover, on page 111](#)
- [Resizing the Docker Storage Backend, on page 112](#)
- [Stopping the Docker Daemon, on page 114](#)
- [Docker Container Security, on page 115](#)
- [Adding Nodes to a Kubernetes Cluster, on page 116](#)
- [Docker Troubleshooting, on page 119](#)

## About Docker with Cisco NX-OS

Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries. See <https://docs.docker.com/> for more information on Docker.

Beginning with Cisco NX-OS Release 9.2(1), support is now added for using Docker within Cisco NX-OS on a switch.

The version of Docker that is included on the switch is CE 18.09.0. The Docker daemon is not running by default. You must start it manually or set it up to automatically restart when the switch boots up.

This section describes how to enable and use Docker in the specific context of the switch environment. Refer to the Docker documentation at <https://docs.docker.com/> for details on general Docker usage and functionality.

## Guidelines and Limitations

Following are the guidelines and limitations for using Docker on Cisco NX-OS on a switch:

- Docker functionality is supported on the switches with at least 8 GB of system RAM.

## Prerequisites for Setting Up Docker Containers Within Cisco NX-OS

Following are the prerequisites for using Docker on Cisco NX-OS on a switch:

- Enable the host Bash shell. To use Docker on Cisco NX-OS on a switch, you must be the root user on the host Bash shell:

```
switch# configure terminal
 Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# feature bash-shell
```

- If the switch is in a network that uses an HTTP proxy server, the `http_proxy` and `https_proxy` environment variables must be set up in `/etc/sysconfig/docker`. For example:

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

- Verify that the switch clock is set correctly, or you might see the following error message:

```
x509: certificate has expired or is not yet valid
```

- Verify that the domain name and name servers are configured appropriately for the network and that it is reflected in the `/etc/resolv.conf` file:

```
switch# conf t
 Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# vrf context management
switch(config-vrf)# ip domain-name ?
WORD Enter the default domain (Max Size 64)

switch(config-vrf)# ip name-server ?
A.B.C.D Enter an IPv4 address
A:B::C:D Enter an IPv6 address

root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch#
```

## Starting the Docker Daemon

When you start the Docker daemon for the first time, a fixed-size backend storage space is carved out in a file called `dockerpart` on the bootflash, which is then mounted to `/var/lib/docker`. If necessary, you can adjust the default size of this space by editing `/etc/sysconfig/docker` before you start the Docker daemon for the first time. You can also resize this storage space if necessary as described later on.

To start the Docker daemon:

## Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start the Docker daemon.

```
root@switch# service docker start
```

**Step 3** Check the status.

```
root@switch# service docker status
dockerd (pid 3597) is running...
root@switch#
```

### Note

Once you start the Docker daemon, do not delete or tamper with the `dockerpart` file on the bootflash since it is critical to the docker functionality.

```
switch# dir bootflash:dockerpart
2000000000 Mar 14 12:50:14 2018 dockerpart
```

# Configure Docker to Start Automatically

You can configure the Docker daemon to always start up automatically when the switch boots up.

## Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Use the `chkconfig` utility to make the Docker service persistent.

```
root@switch# chkconfig --add docker
root@n9k-2#
```

**Step 3** Use the `chkconfig` utility to check the Docker service settings.

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch#
```

**Step 4** To remove the configuration so that Docker does not start up automatically:

```
root@switch# chkconfig --del docker
root@switch# chkconfig --list | grep docker
```

```
root@switch#
```

## Starting Docker Containers: Host Networking Model

If you want Docker containers to have access to all the host network interfaces, including data port and management, start the Docker containers with the `--network host` option. The user in the container can switch between the different network namespaces at `/var/run/netns` (corresponding to different VRFs configured in Cisco NX-OS) using the `ip netns exec <net_namespace> <cmd>`.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start the Docker container.

Following is an example of starting an Alpine Docker container on the switch and viewing all the network interfaces. The container is launched into the management network namespace by default.

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
host --cap-add SYS_ADMIN -it alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
management
default
/ #
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
3: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default
link/gre 0.0.0.0 brd 0.0.0.0
...
/ #
/ # ip netns exec default ip address
```

```

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/16 scope host lo
valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
link/ether 42:0d:9b:3c:d4:62 brd ff:ff:ff:ff:ff:ff
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
...

```

## Starting Docker Containers: Bridged Networking Model

If you want Docker containers to only have external network connectivity (typically through the management interface) and you don't necessarily care about visibility into a specific data port or other switch interface, you can start the Docker container with the default Docker bridged networking model. This is more secure than the host networking model described in the previous section since it also provides network namespace isolation.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start the Docker container.

Following is an example of starting an Alpine Docker container on the switch and installing the `iproute2` package.

```

root@switch# docker run -it --rm alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
/ #

```

**Step 3** Determine if you want to set up user namespace isolation.

For containers using the bridged networking model, you can also set up user namespace isolation to further improve security. See [Securing Docker Containers With User namespace Isolation, on page 115](#) for more information.

You can use standard Docker port options to expose a service from within the container, such as `sshd`. For example:

```
root@switch# docker run -d -p 18877:22 --name sshd_container sshd_ubuntu
```

This maps port 22 from within the container to port 18877 on the switch. The service can now be accessed externally through port 18877, as shown in the following example:

```
root@ubuntu-vm# ssh root@ip_address -p 18887
```

## Mounting the bootflash and volatile Partitions in the Docker Container

You can make the `bootflash` and `volatile` partitions visible in the Docker container by passing in the `-v /bootflash:/bootflash` and `-v /volatile:/volatile` options in the run command for the Docker container. This is useful if the application in the container needs access to files shared with the host, such as copying a new NX-OS system image to bootflash.



**Note** This `-v` command option allows for any directory to be mounted into the container and may result in information leaking or other accesses that may impact the operation of the NX-OS system. Limit this to resources such as `/bootflash` and `/volatile` that are already accessible using NX-OS CLI.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Pass in the `-v /bootflash:/bootflash` and `-v /volatile:/volatile` options in the run command for the Docker container.

```
root@switch# docker run -v /bootflash:/bootflash -v /volatile:/volatile -it --rm alpine
/# ls /
bin etc media root srv usr
bootflash home mnt run sys var
dev lib proc sbin tmp volatile
/ #
```

## Enabling Docker Daemon Persistence on Enhanced ISSU Switchover

You can have both the Docker daemon and any running containers persist on an Enhanced ISSU switchover. This is possible since the bootflash on which the backend Docker storage resides is the same and shared between both Active and Standby supervisors.



The Docker containers are disrupted (restarted) during the switchover, so they will not be running continuously.

## Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Before starting the switchover, use the `chkconfig` utility to make the Docker service persistent.

```
root@switch# chkconfig --add docker
root@n9k-2#
```

**Step 3** Start any containers using the `--restart unless-stopped` option so that they will be restarted automatically after the switchover.

The following example starts an Alpine container and configures it to always restart unless it is explicitly stopped or Docker is restarted:

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

The Docker containers are disrupted (restarted) during the switchover, so they will not be running continuously.

---

# Enabling Docker Daemon Persistence on the Cisco Nexus Platform Switches Switchover

You can have both the Docker daemon and any running containers persist on a switchover between two separate physical supervisors with distinct bootflash partitions. However, for the Cisco Nexus switches, the bootflash partitions on both supervisors are physically separate. You will therefore need to copy the `dockerpart` file manually to the standby supervisor before performing the switchover.

## Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start any containers using the `--restart unless-stopped` option so that they will be restarted automatically after the switchover.

The following example starts an Alpine container and configures it to always restart unless it is explicitly stopped or Docker is restarted:

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

Note that the Docker containers will be disrupted (restarted) during the switchover, so they will not be running continuously.

**Step 3** Before starting the switchover, use the `chkconfig` utility to make the Docker service persistent.

```
root@switch# chkconfig --add docker
root@n9k-2#
```

**Step 4** Copy the Docker backend storage partition from the active to the standby supervisor bootflash:

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown

root@switch# cp /bootflash/dockerpart /bootflash_sup-remote/

root@switch# service docker start
```

## Resizing the Docker Storage Backend

After starting or using the Docker daemon, you can grow the size of the Docker backend storage space according to your needs.

### Procedure

**Step 1** Disable the Guest Shell.

If you do not disable the Guest Shell, it may interfere with the resize.

```
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
switch# 2018 Mar 15 17:16:55 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2018 Mar 15 17:16:57 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated virtual
service 'guestshell+'
```

**Step 2** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 3** Get information on the current amount of storage space available.

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 1.9G 7.6M 1.8G 1% /var/lib/docker
root@n9k-2#
```

**Step 4** Stop the Docker daemon.

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

**Step 5** Get information on the current size of the Docker backend storage space (`/bootflash/dockerpart`).

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2000000000 Mar 15 16:53 /bootflash/dockerpart
root@n9k-2#
```

**Step 6** Resize the Docker backend storage space.

For example, the following command increases the size by 500 megabytes:

```
root@switch# truncate -s +500MB /bootflash/dockerpart
root@n9k-2#
```

**Step 7** Get updated information on the size of the Docker backend storage space to verify that the resizing process was completed successfully.

For example, the following output confirms that the size of the Docker backend storage was successfully increased by 500 megabytes:

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2500000000 Mar 15 16:54 /bootflash/dockerpart
root@n9k-2#
```

**Step 8** Check the size of the filesystem on /bootflash/dockerpart.

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/122160 files (0.6% non-contiguous), 17794/488281 blocks
```

**Step 9** Resize the filesystem on /bootflash/dockerpart.

```
root@switch# /sbin/resize2fs /bootflash/dockerpart
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /bootflash/dockerpart to 610351 (4k) blocks.
The filesystem on /bootflash/dockerpart is now 610351 blocks long.
```

**Step 10** Check the size of the filesystem on /bootflash/dockerpart again to confirm that the filesystem was successfully resized.

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/154736 files (0.6% non-contiguous), 19838/610351 blocks
```

**Step 11** Start the Docker daemon again.

```
root@switch# service docker start
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Starting dockerd with args '--debug=true':
```

**Step 12** Verify the new amount of storage space available.

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 2.3G 7.6M 2.3G 1% /var/lib/docker
```

**Step 13** Exit out of Bash shell.

```
root@switch# exit
logout
switch#
```

**Step 14** Enable the Guest Shell, if necessary.

```
switch# guestshell enable

switch# 2018 Mar 15 17:12:53 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
'guestshell+'
switch# 2018 Mar 15 17:13:18 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```

## Stopping the Docker Daemon

If you no longer wish to use Docker, follow the procedures in this topic to stop the Docker daemon.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Stop the Docker daemon.

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

**Step 3** Verify that the Docker daemon is stopped.

```
root@switch# service docker status
dockerd is stopped
root@switch#
```

#### Note

You can also delete the `dockerpart` file on the bootflash at this point, if necessary:

```
switch# delete bootflash:dockerpart
Do you want to delete "/dockerpart" ? (yes/no/abort) y
```

```
switch#
```

---

## Docker Container Security

Following are the Docker container security recommendations:

- Run in a separate user `namespace` if possible.
- Run in a separate network `namespace` if possible.
- Use `cgroups` to limit resources. An existing `cgroup` (`ext_ser`) is created to limit hosted applications to what the platform team has deemed reasonable for extra software running on the switch. Docker allows use of this and limiting per-container resources.
- Do not add unnecessary POSIX capabilities.

## Securing Docker Containers With User namespace Isolation

For containers using the bridged networking model, you can also set up user namespace isolation to further improve security. See <https://docs.docker.com/engine/security/usersns-remap/> for more information.

### Procedure

---

**Step 1** Determine if a `dockremap` group already exists on your system.

A `dockremap` user must already be set up on your system by default. If the `dockremap` group doesn't already exist, follow these steps to create it.

a) Enter the following command to create the `dockremap` group:

```
root@switch# groupadd dockremap -r
```

b) Create the `dockremap` user, unless it already exists:

```
root@switch# useradd dockremap -r -g dockremap
```

c) Verify that the `dockremap` group and the `dockremap` user were created successfully:

```
root@switch# id dockremap
uid=999(dockremap) gid=498(dockremap) groups=498(dockremap)
root@switch#
```

**Step 2** Add the desired re-mapped ID and range to the `/etc/subuid` and `/etc/subgid`.

For example:

```
root@switch# echo "dockremap:123000:65536" >> /etc/subuid
root@switch# echo "dockremap:123000:65536" >> /etc/subgid
```

**Step 3** Using a text editor, add the `--usersns-remap=default` option to the `other_args` field in the `/etc/sysconfig/docker` file.

For example:

```
other_args="--debug=true --usersns-remap=default"
```

**Step 4** Restart the Docker daemon, or start it if it is not already running, using `service docker [re]start`.

For example:

```
root@switch# service docker [re]start
```

Refer to the Docker documentation at <https://docs.docker.com/engine/security/usersns-remap/> for more information on configuring and using containers with user namespace isolation.

## Moving the cgroup Partition

The cgroup partition for third-party services is `ext_ser`, which limits CPU usage to 25% per core. Cisco recommends that you run your Docker container under this `ext_ser` partition.

If the Docker container is run without the `--cgroup-parent=/ext_ser/` option, it can get up to the full 100% host CPU access, which can interfere with the regular operation of Cisco NX-OS.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Run the Docker container under the `ext_ser` partition.

For example:

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
host --cgroup-parent=/ext_ser/ --cap-add SYS_ADMIN -it alpine
/ #
```

## Adding Nodes to a Kubernetes Cluster

This topic describes how to add nodes to a Kubernetes cluster. In this example:

- The Kubernetes (Ubuntu) primary has an IP address of 10.122.197.246
- The switch software running as Docker containers has an IP address of 10.122.84.24



**Note** In the following examples, long single lines of text are broken up with the \ character to improve readability.

## Procedure

**Step 1** Run the following commands (on? for?) the Kubernetes (Ubuntu) primary.

a) Enter this command:

```
root@switch# docker run -d --net=host gcr.io/google_containers/etcd:2.2.1 /usr/local/bin/etcd
--listen-client-urls=http://0.0.0.0:4001 --advertise-client-urls=http://0.0.0.0:4001
--data-dir=/var/etcd/data
```

b) Enter this command:

```
root@switch# docker run -d --name=api --net=host --pid=host --privileged=true
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube apiserver --insecure-bind-address=0.0.0.0
--allow-privileged=true --service-cluster-ip-range=10.0.0.1/24 --etcd_servers=http://127.0.0.1:4001
--v=2
```

c) Enter this command:

```
root@switch# docker run -d --name=kubs --volume=:/rootfs:ro --volume=/sys:/sys:ro
--volume=/dev:/dev --volume=/var/lib/docker:/var/lib/docker:rw
--volume=/var/lib/kubelet:/var/lib/kubelet:rw --volume=/var/run:/var/run:rw --net=host --pid=host
--privileged=true gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube kubelet
--allow-privileged=true --hostname-override="127.0.0.1" --address="0.0.0.0"
--api-servers=http://0.0.0.0:8080 --cluster_dns=10.0.0.10 --cluster_domain=cluster.local
--config=/etc/kubernetes/manifests-multi
```

d) Enter this command:

```
root@switch# docker run -d --name=proxy --net=host --privileged
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube proxy --master=http://0.0.0.0:8080 --v=2
```

e) Enter this command:

```
root@switch# export KUBERNETES_MASTER=http://10.122.197.246:8080
```

f) Enter this command:

```
root@switch# curl -o /usr/bin/kubectl
http://storage.googleapis.com/kubernetes-release/release/v1.2.2/bin/linux/amd64/kubectl
```

g) Enter this command:

```
root@switch# kubectl -s $KUBERNETES_MASTER create -f kube-system.json
```

h) Enter this command:

```
root@switch# kubectl -s $KUBERNETES_MASTER create -f skydns-rc.yaml
```

i) Enter this command:

```
root@switch# kubectl -s $KUBERNETES_MASTER create -f skydns-svc.yaml
```

- j) Enter this command:

```
root@switch# kubectl -s $KUBERNETES_MASTER create -f dashboard.yaml
kubectl -s $KUBERNETES_MASTER cluster-info
```

- k) Enter this command:

```
root@switch# kubectl -s $KUBERNETES_MASTER cluster-info
```

## Step 2 Run the following steps (on? for?) the switch.

- a) Enter this command:

```
root@switch# docker run -d --name=kubs --net=host --pid=host --privileged=true --volume=/:/rootfs:ro
--volume=/sys:/sys:ro --volume=/dev:/dev --volume=/var/lib/docker/:/var/lib/docker:rw
--volume=/var/lib/kubelet/:/var/lib/kubelet:rw --volume=/var/run:/var/run:rw
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube kubelet --allow-privileged=true --containerized
--enable-server --cluster_dns=10.0.0.10 --cluster_domain=cluster.local
--config=/etc/kubernetes/manifests-multi --hostname-override="10.122.84.34" --address=0.0.0.0
--api-servers=http://10.122.197.246:8080
```

- b) Enter this command:

```
root@switch# docker run -d --name=proxy --net=host --privileged=true
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube proxy --master=http://10.122.197.246:8080
--v=2
```

## Step 3 Run the following commands (on? for?) the Kubernetes (Ubuntu) primary to deploy an nginx app in a replication controller object.

- a) Enter this command:

```
lab@rnimbalk-ubuntu1:~$ kubectl get node
NAME STATUS AGE
10.122.84.34 Ready 16m
127.0.0.1 Ready 22m
```

- b) Enter this command:

```
lab@rnimbalk-ubuntu1:~$ kubectl apply -f replication.yaml
replicationcontroller "nginx" created
```

- c) Enter this command:

```
lab@rnimbalk-ubuntu1:~$ kubectl describe -f replication.yaml
Name: nginx
Namespace: default
Image(s): nginx
Selector: app=nginx
Labels: app=nginx
Replicas: 3 current / 3 desired
Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
 FirstSeen LastSeen Count From SubobjectPath Type
 Reason Message
```





```
root@switch# service docker start
Free bootflash: 790 MB, total bootflash: 3471 MB
Need at least 2000 MB free bootflash space for docker storage
```

**Possible Cause:** You might not have enough free bootflash storage.

**Solution:** Free up space or adjust the `variable_dockerstrg` values in `/etc/sysconfig/docker` as needed, then restart the Docker daemon:

```
root@switch# cat /etc/sysconfig/docker
Replace the below with your own docker storage backend boundary value (in MB)
if desired.
boundary_dockerstrg=5000

Replace the below with your own docker storage backend values (in MB) if
desired. The smaller value applies to platforms with less than
$boundary_dockerstrg total bootflash space, the larger value for more than
$boundary_dockerstrg of total bootflash space.
small_dockerstrg=300
large_dockerstrg=2000
```

## Failure to Pull Images from Docker Hub (509 Certificate Expiration Error Message)

**Problem:** The system fails to pull images from the Docker hub with an error message similar to the following:

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: x509: certificate has
expired or is not yet valid
```

**Possible Cause:** The system clock might not be set correctly.

**Solution:** Determine if the clock is set correctly or not:

```
root@n9k-2# sh clock
15:57:48.963 EST Thu Apr 25 2002
Time source is Hardware Calendar
```

Reset the clock, if necessary:

```
root@n9k-2# clock set hh:mm:ss { day month | month day } year
```

For example:

```
root@n9k-2# clock set 14:12:00 10 feb 2018
```

## Failure to Pull Images from Docker Hub (Client Timeout Error Message)

**Problem:** The system fails to pull images from the Docker hub with an error message similar to the following:

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled
while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

**Possible Cause:** The proxies or DNS settings might not be set correctly.

**Solution:** Check the proxy settings and fix them, if necessary, then restart the Docker daemon:

```
root@switch# cat /etc/sysconfig/docker | grep proxy
#export http_proxy=http://proxy.esl.cisco.com:8080
#export https_proxy=http://proxy.esl.cisco.com:8080
root@switch# service docker [re]start
```

Check the DNS settings and fix them, if necessary, then restart the Docker daemon:

```
root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch# # conf t
 Enter configuration commands, one per line. End with CNTL/Z.
 switch(config)# vrf context management
 switch(config-vrf)# ip domain-name ?
 WORD Enter the default domain (Max Size 64)

 switch(config-vrf)# ip name-server ?
 A.B.C.D Enter an IPv4 address
 A:B::C:D Enter an IPv6 address
root@switch# service docker [re]start
```

## Docker Daemon or Containers Not Running On Switch Reload or Switchover

**Problem:** The Docker daemon or containers do not run after you have performed a switch reload or switchover.

**Possible Cause:** The Docker daemon might not be configured to persist on a switch reload or switchover.

**Solution:** Verify that the Docker daemon is configured to persist on a switch reload or switchover using the `chkconfig` command, then start the necessary Docker containers using the `--restart unless-stopped` option. For example, to start an Alpine container:

```
root@switch# chkconfig --add docker
root@switch#
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch# docker run -dit --restart unless-stopped alpine
```

## Resizing of Docker Storage Backend Fails

**Problem:** An attempt to resize the Docker backend storage failed.

**Possible Cause:** You might not have Guest Shell disabled.

**Solution:** Use the following command to determine if Guest Shell is disabled:

```
root@switch# losetup -a | grep dockerpart
root@n9k-2#
```

The command should not display any output if Guest Shell is disabled.

Enter the following command to disable the Guest Shell, if necessary:

```
switch# guestshell disable
```

If you still cannot resize the Docker backend storage, you can delete `/bootflash/dockerpart`, then adjust the `[small_]large_dockerstrg` in `/etc/sysconfig/docker`, then start Docker again to get a fresh Docker partition with the size that you want.

## Docker Container Doesn't Receive Incoming Traffic On a Port

**Problem:** The Docker container doesn't receive incoming traffic on a port.

**Possible Cause:** The Docker container might be using a netstack port instead of a kstack port.

**Solution:** Verify that any ephemeral ports that are used by Docker containers are within the kstack range. Otherwise any incoming packets can get sent to netstack for servicing and dropped.

```
switch# show socket local-port-range
Kstack local port range (15001 - 58000)
Netstack local port range (58001 - 63535) and nat port range (63536 - 65535)
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# sockets local-port-range <start_port> <end_port>
switch# run bash sudo su -
root@switch# cat /proc/sys/net/ipv4/ip_local_port_range
15001 58000
root@switch#
```

## Unable to See Data Port And/Or Management Interfaces in Docker Container

**Problem:** You are unable to see the data port or management interfaces in the Docker container.

**Solution:**

- Verify that the Docker container is started in the host network namespace with all host namespaces mapped in using the `-v /var/run/netns:/var/run/netns:ro,rslave --network host` options.
- Once in the container, you will be in the management network namespace by default. You can use the `ip netns` utility to move to the default (`init`) network namespace, which has the data port interfaces. The `ip netns` utility might need to be installed in the container using `dnf`, `apk`, or something similar.

## General Troubleshooting Tips

**Problem:** You have other issues with Docker containers that were not resolved using other troubleshooting processes.

**Solution:**

- Look for `dockerd` debug output in `/var/log/docker` for any clues as to what is wrong.
- Verify that your switch has 8 GB or more of RAM. Docker functionality is not supported on any switch that has less than 8 GB of RAM.



# CHAPTER 13

## NX-SDK

---

This chapter contains the following sections:

- [About the NX-SDK, on page 123](#)
- [About On-Box \(Local\) Applications, on page 124](#)
- [Default Docker Images, on page 124](#)
- [Guidelines and Limitations for NX-SDK, on page 125](#)
- [Off-Box \(Remote\) Applications, on page 126](#)

## About the NX-SDK

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

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

You can use C++, Python, or Go (with some exceptions) for application development with NX-SDK.

### Requirements

The NX-SDK has the following requirements:

- Docker
- A Linux environment (either Ubuntu 14.04, or Centos 6.7). Cisco recommends using the provided NX-SDK Docker containers. For more information, see [Cisco DevNet NX-SDK](#).

### Support for Local (On Switch) and Remote (Off Switch) Applications

Applications that are developed with NX-SDK are created or developed off of the switch in the Docker containers that the NX-SDK provides. After the application is created, you have flexibility of where the applications can be deployed:

- Local (on-box) applications run on the switch. For information, see [About On-Box \(Local\) Applications, on page 124](#)
- Remote (off-box) applications run off switch. This option, supported with NX-SDK 2.0 and later, enables you to deploy the application to run anywhere other than on the switch. For information, see [About Remote Applications, on page 126](#).

### Related Information

For more information about Cisco NX-SDK, go to:

- [Cisco DevNet NX-SDK](#). Click the `versions.md` link ([Cisco DevNet NX-SDK Versions](#)) to get information about features and details on each supported release.
- [NX-SDK Readmes](#)

As needed, Cisco will add information for NX-SDK to github.

## Considerations for Go Bindings

Go bindings are supported at various levels depending on the release of NX-SDK and whether apps are running locally or remotely.

- Go bindings for any version of NX-SDK remote application are pre-EFT quality.
- Go bindings for a local NX-SDK 2.0 application is pre-EFT.
- Go bindings for a local NX-SDK 1.7.5 application or earlier is supported.

For more information, see [GO Bindings for NX-SDK Applications](#).

## About On-Box (Local) Applications

With on box (local) applications, you install the NX-SDK, build your application in whichever supported language you choose, package the app as an `.rpm` file which can be installed on the switch, then install and run your applications on the switch. The `.rpm` files can be manually generated or autogenerated.

Application development occurs in the containers that are provided by NX-SDK. You will use a different container and tools for local applications than remote applications. For more information, see [Default Docker Images, on page 124](#).

For information about building, installing, and running local applications, see [Cisco DevNet NX-SDK](#).

## Default Docker Images

NX-SDK has the following Docker images and tools by default for local or remote use.

Usage	Contents
On Switch	Cisco ENXOS SDK Wind River Linux (WRL) tool chain for cross compiling Multi-language binding toolkit Beginning with NX-SDK 1.75, a Go compiler
Off switch (remote)	NX-SDK multi-language binding Toolkit with pre-built libnxsdk.so A Go compiler RapidJSON gRPC for remote API support

For more information, see <https://github.com/CiscoDevNet/NX-SDK#readme>[https://github.com/CiscoDevNet/NX-SDK/tree/master/readmes/nxsdk\\_docker\\_images.md](https://github.com/CiscoDevNet/NX-SDK/tree/master/readmes/nxsdk_docker_images.md).

## Guidelines and Limitations for NX-SDK

NX-SDK has usage guidelines and limitations for running applications locally (on box) or remotely (off box).

For guidelines and limitations, see "Helpful Notes" at [Cisco DevNet NX-SDK](#).

- For remote applications, ports that connect to the SDK server are from 50002 through 50100. Make sure that these ports are open and not used by other services. Port 50051 is blocked for use by an internal application, and cannot be used by remote applications.
- The following limitations apply to developing a custom application through NX-SDK:
  - You must develop applications with any current Linux distribution that have GNU C/C++ toolchains and standard libraries.
  - We recommend developing applications in the provided Docker images. Run the Docker image and associated tools requires a minimum of 8 MB of free memory and 64-bit hosts. See [Default Docker Images, on page 124](#).
- Cisco recommends that you build and test your applications in Bash. When you deploy them in production, build the applications as RPMs and run them in the NX-OS VSH (Vshell).
- Cisco Nexus switches support a maximum of 32 applications in Bash and VSH.
- For NX-SDK 2.0, there is a 1-to-1 limit for remote NX-SDK applications and NX-SDK servers running in the Cisco Nexus switch.
- A maximum of 10 remote NX-SDK applications can run simultaneously in a switch.
- The NX-SDK server accepts remote requests only from a CLI-configured application and not from random applications.
- For any error that occurs, the SDK server throws an exception.
- Some APIs do not run in a remote application. Refer to API documentation for more information.

- Remote client library code is not open source, so NX-SDK remote apps need to run in the NX-SDK remote docker container. To use remote NX-SDK in your own OS, make a request in github with your OS and compiler version. Cisco can generate a remote libnxsdk.so based on your version.

## Off-Box (Remote) Applications

### About NX-SDK 2.0

The NX-SDK version 2.0 enables execution-environment flexibility for developers to run their applications wherever needed. With this version of NX-SDK, your applications are still developed off the switch in containers, but you can run the apps either on the switch or off the switch, for example in a cloud.

NX-SDK 2.0 offers the following benefits:

- Easy integration of the switch into the customer environment.
- Scalability to enable the switch to seamlessly operate in data centers, public clouds, and private clouds.
- Decoupling customer apps from switch resources so that changes at the switch-level resources do not require change or rewrite of applications.
- Single library with simple to use APIs for applications to link against, which simplifies switch interactions and allows applications to be written in high-level languages that are easier to write and debug.
- Running Remote services are more secure than on-box applications.

For more information, see [https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK\\_in\\_NXOS.md](https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md) Deploying NX-SDK Applications Remotely.

### About Remote Applications

Remote applications can be on a different switch that is not a Cisco Nexus switch. Remote, or off-box, applications call through the NX-SDK layer to interact with the switch to read information (get) or write information (set).

Both local and remote NX-SDK applications use the same APIs, which offer you the flexibility to deploy NX-SDK applications on- or off-box.

To run remotely, an application must meet specific requirements. For information, see [https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK\\_in\\_NXOS.md](https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md) Deploying NX-SDK Applications Remotely.

#### Backward Compatibility for Pre-2.0 NX-SDK Applications

NX-SDK 2.0 has conditional backward compatibility for NX-SDK v1.75 applications depending on how these applications were developed:

- Usually, NX-SDK supports remotely running an app that you created before NX-SDK 2.0 without requiring you to completely rewrite your app. Instead, you can reuse the same app without modifying it to change the API calls. To support older apps in the new NX-SDK 2.0 model, the API call must provide IP and Port parameters. These parameters are not available in NX-SDK 1.75 and earlier, but you can add the IP address and Port information as environment variables that the app can export to the SDK server.



- However, sometimes backward compatibility for pre-NX-SDK 2.0 apps might not be supported. It is possible that some APIs in older apps might not support, or be capable of, running remotely. In this case, the APIs can throw an exception. Depending on how complete and robust the exception-handling is for the original application, the application might operate unpredictably, and in worst cases, possibly crash.

For more information, see [https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK\\_in\\_NXOS.md](https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md) **Deploying NX-SDK Applications Remotely**.

## NX-SDK Security

Beginning with NX-OS 9.3(1), NX-SDK 2.0 supports the following security features:

- **Session security.** Remote applications can connect to the NX SDK server on the switch through Transport Layer Service (TLS) to provide encrypted sessions between the applications and the switch's NX SDK server.
- **Server certificate security.** For new switch deployments with Cisco NX-OS 9.3(1), the NX-SDK server generates a one-day temporary certificate to provide enough time to install a custom certificate.

If your NX-SDK server already has a custom certificate that is installed, for example, if you are upgrading from a previous NX-SDK version to NX-SDK 2.0, your existing certificate is retained and used after upgrade.

- **API write-call control.** NX-SDK 2.0 introduces security profiles, which enable you to select a pre-defined policy for controlling how much control an application has with the NX-SDK server. For more information about security profiles, see [Security Profiles for NX SDK 2.0, on page 127](#).

## Security Profiles for NX SDK 2.0

In previous releases, the APIs for SDK version 1.75 were permitted only to read and get data for events. Beginning in Cisco NX-OS Release 9.3(1), NX-SDK 2.0 supports different types of operations, including write calls.

The ability of an app to read or write to the switch can be controlled through a security profile. A security profile is an optional object that is attached to the applications' service running in the switch. Security profiles control an application's ability to write to the switch, and in turn, control the applications ability to modify, delete, or configure switch functionality. By default, application writes are disallowed, so for each application, you will need to create a security profile that enables write access to the switch.

Cisco's NX-SDK offers the following security profiles.

Profile	Description	Values
Deny	Prevents any API calls from writing to the switch except for adding CLIs.	This is the default profile.

Profile	Description	Values
Throttle	<p>Allows APIs that modify the switch, but only up to a specified number of calls. This security profile applies throttling to control the number of API calls.</p> <p>The application is allowed to write up to the limit, but when the limit is exceeded, writing stops, and the reply sends an error message.</p>	The throttle is 50 API calls, and the throttle resets after five seconds.
Permit	APIs that modify the switch are allowed without restriction	

For more information about security profiles in NX-SDK, see [Security Profiles for NX-SDK Applications](#) .  
 For additional information about building, installing, and running applications, go to [CiscoDevNet NX-SDK](#) .



# CHAPTER 14

## NX-API

---

- [About NX-API, on page 129](#)
- [Using NX-API, on page 131](#)
- [XML and JSON Supported Commands, on page 148](#)
- [JSON-RPC, JSON, and XML Supported Commands, on page 156](#)

### About NX-API

On Cisco Nexus switches, command-line interfaces (CLIs) are run only on the switch. NX-API improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco NX-OS CLI system on the Cisco Nexus 7000 Series switches. NX-API supports **show** commands, and configurations.

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

On Cisco Nexus switches, command-line interfaces (CLIs) are run only on the switch. NX-API improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco NX-OS CLI system on the Cisco Nexus 9000 Series switches. NX-API supports **show** commands, configurations, and Linux Bash.

NX-API supports JSON-RPC.

On Cisco Nexus switches, command-line interfaces (CLIs) are run only on the switch. NX-API improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco NX-OS CLI system on the Cisco Nexus 3500 platform switches. NX-API supports **show** commands, configurations, and Linux Bash.

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

### Feature NX-API

- Feature NX-API is required to be enabled for access the device through sandbox.
- | json on the device internally uses python script to generate output.
- NX-API can be enabled either on http/https via ipv4:

```
BLR-VXLAN-NPT-CR-179# show nxapi
nxapi enabled
HTTP Listen on port 80
```

```
HTTPS Listen on port 443
BLR-VXLAN-NPT-CR-179#
```

- NX-API is internally spawning third-party NGINX process, which handler receive/send/processing of http requests/response:

```
nxapi certificate {httpsCRT |httpskey}
nxapi certificate enable
```

- NX-API Certificates can be enabled for https
- Default port for nginx to operate is 80/443 for http/https respectively. It can also be changed using the following CLI command:

```
nxapi {http|https} port port-number
```

## Transport

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

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




---

**Note** The Nginx process continues to run even after NX-API is disabled using the `no feature NXAPI` command. This is required for other management-related processes.

---

## Message Format

NX-API is an enhancement to the Cisco Nexus 7000 Series CLI system, which supports XML output. NX-API also supports JSON output format for specific commands.

NX-API is an enhancement to the Cisco Nexus 9000 Series CLI system, which supports XML output. NX-API also supports JSON output format for specific commands.




---

**Note**

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

---

## Security

NX-API supports HTTPS. All communication to the device is encrypted when you use HTTPS.

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



---

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

---

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

NX-API provides a session-based cookie, **nxapi\_auth** when users first successfully authenticate. With the session cookie, the username and password are included in all subsequent NX-API requests that are sent to the device. The username and password are used with the session cookie to bypass performing the full authentication process again. If the session cookie is not included with subsequent requests, another session cookie is required and is provided by the authentication process. Avoiding unnecessary use of the authentication process helps to reduce the workload on the device.



---

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

When the cookie expires, you need to resend your user name/password.

---



---

**Note** NX-API performs authentication through a programmable authentication module (PAM) on the switch. Use cookies to reduce the number of PAM authentications, which reduces the load on the PAM.

---

## Using NX-API

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

The commands, command type, and output type for the Cisco Nexus 9000 Series switches are entered using NX-API by encoding the CLIs into the body of a HTTP/HTTPS POST. The response to the request is returned in XML or JSON output format.

The commands, command type, and output type for the Cisco Nexus 3500 platform switches are entered using NX-API by encoding the CLIs into the body of a HTTP/HTTPS POST. The response to the request is returned in XML, JSON, or JSON-RPC output format.



---

**Note** For more details about NX-API response codes, see e [Table of NX-API Response Codes](#).

---

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

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

- Enable the management interface.

```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 198.51.100.1/24
```

```
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

- Enable the NX-API `nxapi` feature.

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

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

#### Request:

```
<?xml version="1.0" 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"
 }
 }
 }
 }
 }
}

```

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"
 }
 }
 }
}

```

### Using the Management Interface for NX-API calls

It is recommended to use the management interface for NX-API calls.

When using non-management interface and a custom port for NX-API an entry should be made in the CoPP policy to prevent NX-API traffic from hitting the default copp entry which could unfavorably treat API traffic.




---

**Note** It is recommended to use the management interface for NX-API traffic. If that is not possible and a custom port is used, the "copp-http" class should be updated to include the custom NX-API port.

---

The following example port 9443 is being used for NX-API traffic.

This port is added to the copp-system-acl-http ACL to allow it to be matched under the copp-http class resulting on 100 pps policing. (This may need to be increased in certain environments.)

```

!
ip access-list copp-system-acl-http
 10 permit tcp any any eq www
 20 permit tcp any any eq 443
 30 permit tcp any any eq 9443 <-----
!
class-map type control-plane match-any copp-http
 match access-group name copp-system-acl-http
!
policy-map type control-plane copp-system-policy
 class copp-http
 police pps 100
!

```

## NX-API Management Commands

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



Table 7: NX-API Management Commands

NX-API Management Command	Description
<b>feature nxapi</b>	Enables NX-API.
<b>no feature nxapi</b>	Disables NX-API.
<b>nxapi {http   https} port <i>port</i></b>	Specifies a port.
<b>no nxapi {http   https}</b>	Disables HTTP/HTTPS.
<b>show nxapi</b>	Displays port information.
<b>nxapi certificate {httpsrct certfile   httpskey keyfile} <i>filename</i></b>	Specifies the upload of the following: <ul style="list-style-type: none"> <li>• HTTPS certificate when <code>httpsrct</code> is specified.</li> <li>• HTTPS key when <code>httpskey</code> is specified.</li> </ul> <p>Example of HTTPS certificate:</p> <pre>nxapi certificate httpsrct certfile bootflash:cert.crt</pre> <p>Example of HTTPS key:</p> <pre>nxapi certificate httpskey keyfile bootflash:privkey.key</pre>
<b>nxapi certificate enable</b>	Enables a certificate.

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

```
switch(config)# nxapi certificate httpsrct 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 might 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
Nginx certificate invalid.
switch(config)#
```

This might occur if the key file is encrypted. In that case, the key file must be decrypted 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)#
```

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 timing out 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 where **terminal dont-ask** is used 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 request elements are sent to the switch in XML format, JSON format, or JSON-RPC format. The HTTP header of the request must identify the content type of the request.

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

**Table 8: NX-API Request Elements**

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

NX-API Request Element	Description
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> <li>• <b>cli_show</b> CLI <b>show</b> commands that expect structured output. If the command does not support XML output, an error message is returned.</li> <li>• <b>cli_show_ascii</b> CLI <b>show</b> commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes.</li> <li>• <b>cli_conf</b> CLI configuration commands.</li> <li>• <b>bash</b> Bash commands. Most non-interactive Bash commands are supported by NX-API.</li> </ul> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Each command is only executable with the current user's authority.</li> <li>• 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.</li> <li>• A maximum of 10 consecutive <b>show</b> commands are supported. If the number of <b>show</b> commands exceeds 10, the 11th and subsequent commands are ignored.</li> <li>• No interactive commands are supported.</li> </ul>

NX-API Request Element	Description				
<i>chunk</i>	<p>Some <b>show</b> 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 <b>show</b> commands.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="786 485 1479 596"> <tr> <td data-bbox="786 485 899 539">0</td> <td data-bbox="899 485 1479 539">Do not chunk output.</td> </tr> <tr> <td data-bbox="786 539 899 596">1</td> <td data-bbox="899 539 1479 596">Chunk output.</td> </tr> </table> <p><b>Note</b> 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 &lt; or &gt;, are converted to form a valid XML message (&lt; is converted into &amp;lt; &gt; is converted into &amp;gt;).</p> <p>You can use XML SAX to parse the chunked output.</p> <p><b>Note</b> When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</p> <p><b>Note</b> When chunking is enabled, the maximum message size supported is currently 200MB of chunked output.</p>	0	Do not chunk output.	1	Chunk output.
0	Do not chunk output.				
1	Chunk output.				
<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> <p>Cisco NX-OS release 9.3(1) introduces the <i>sid</i> option <code>clear</code>. When a new chunk request is initiated with the <i>sid</i> set to <code>clear</code>, all current chunk requests are discarded or abandoned.</p> <p>When you receive response code 429: Max number of concurrent chunk request is 2, use <i>sid clear</i> to abandon the current chunk requests. After using <i>sid clear</i>, subsequent response codes operate as usual per the rest of the request.</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, <b>show</b> commands are cli_show message type and are not supported in cli_conf mode.</p> <p><b>Note</b> Except for <b>bash</b>, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.) For <b>bash</b>, multiple commands are separated with ";". (The ; is <b>not</b> surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="824 688 1523 915"> <tbody> <tr> <td data-bbox="824 688 948 764">cli_show</td> <td data-bbox="948 688 1523 764">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="824 764 948 840">cli_conf</td> <td data-bbox="948 764 1523 840">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="824 840 948 915">bash</td> <td data-bbox="948 840 1523 915">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="824 989 1523 1104"> <tbody> <tr> <td data-bbox="824 989 1062 1045">xml</td> <td data-bbox="1062 989 1523 1045">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="824 1045 1062 1104">json</td> <td data-bbox="1062 1045 1523 1104">Specifies output in JSON format.</td> </tr> </tbody> </table> <p><b>Note</b> The Cisco Nexus 7000 Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.</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 9: NX-API Request Elements**

NX-API Request Element	Description
<i>jsonrpc</i>	<p>A String specifying the version of the JSON-RPC protocol.</p> <p>Version must be 2.0.</p>

NX-API Request Element	Description
<i>method</i>	<p>A string containing the name of the method to be invoked.</p> <p>NX-API supports either:</p> <ul style="list-style-type: none"> <li>• <code>cli show</code> or configuration commands</li> <li>• <code>cli_ascii show</code> or configuration commands; output without formatting</li> </ul>
<i>params</i>	<p>A structured value that holds the parameter values used during the invocation of the method.</p> <p>It must contain the following:</p> <ul style="list-style-type: none"> <li>• <code>cmd</code> a CLI command</li> <li>• <code>version</code> NX-API request version identifier</li> </ul>
<i>id</i>	<p>An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should normally not be null and numbers contain no fractional parts. If the user does not specify the <i>id</i> parameter, the server assumes the request is simply a notification resulting in a no response. For example, <i>id</i> : 1</p>

NX-API request elements are sent to the switch 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 10: NX-API Request Elements**

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

NX-API Request Element	Description
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> <li>• <b>cli_show</b> CLI <b>show</b> commands that expect structured output. If the command does not support XML output, an error message is returned.</li> <li>• <b>cli_show_ascii</b> CLI <b>show</b> commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes.</li> <li>• <b>cli_conf</b> CLI configuration commands.</li> <li>• <b>bash</b> Bash commands. Most non-interactive Bash commands are supported by NX-API.</li> </ul> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Each command is only executable with the current user's authority.</li> <li>• 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.</li> <li>• A maximum of 10 consecutive <b>show</b> commands are supported. If the number of <b>show</b> commands exceeds 10, the 11th and subsequent commands are ignored.</li> <li>• No interactive commands are supported.</li> </ul>

NX-API Request Element	Description				
<i>chunk</i>	<p>Some <b>show</b> 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 <b>show</b> commands.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="786 485 1481 596"> <tr> <td data-bbox="786 485 899 541">0</td> <td data-bbox="899 485 1481 541">Do not chunk output.</td> </tr> <tr> <td data-bbox="786 541 899 596">1</td> <td data-bbox="899 541 1481 596">Chunk output.</td> </tr> </table> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Only <b>show</b> commands support chunking. When a series of <b>show</b> commands are entered, only the first command is chunked and returned.</li> <li>• The output message format options are XML or JSON.</li> <li>• For the XML output message format, special characters, such as &lt; or &gt;, are converted to form a valid XML message (&lt; is converted into &amp;lt; &gt; is converted into &amp;gt;).</li> </ul> <p>You can use XML SAX to parse the chunked output.</p> <ul style="list-style-type: none"> <li>• When the output message format is JSON, the chunks are concatenated to create a valid JSON object.</li> </ul>	0	Do not chunk output.	1	Chunk output.
0	Do not chunk output.				
1	Chunk output.				
<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> <p>NX-OS release 9.3(1) introduces the <i>sid</i> option <code>clear</code>. When a new chunk request is initiated with the <i>sid</i> set to <code>clear</code>, all current chunk requests are discarded or abandoned.</p> <p>When you receive response code 429: Max number of concurrent chunk request is 2, use <i>sid clear</i> to abandon the current chunk requests. After using <i>sid clear</i>, subsequent response codes operate as usual per the rest of the request.</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, <b>show</b> commands are cli_show message type and are not supported in cli_conf mode.</p> <p><b>Note</b> Except for <b>bash</b>, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.) For <b>bash</b>, multiple commands are separated with ";". (The ; is <b>not</b> surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="824 688 1523 919"> <tbody> <tr> <td data-bbox="824 688 948 764">cli_show</td> <td data-bbox="948 688 1523 764">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="824 764 948 840">cli_conf</td> <td data-bbox="948 764 1523 840">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="824 840 948 915">bash</td> <td data-bbox="948 840 1523 915">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="824 991 1523 1104"> <tbody> <tr> <td data-bbox="824 991 1062 1045">xml</td> <td data-bbox="1062 991 1523 1045">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="824 1045 1062 1104">json</td> <td data-bbox="1062 1045 1523 1104">Specifies output in JSON format.</td> </tr> </tbody> </table> <p><b>Note</b> The Cisco Nexus 9000 Series CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.</p> <p>To manage the computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported.</p> <p>The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).</p>	xml	Specifies output in XML format.	json	Specifies output in JSON format.		
xml	Specifies output in XML format.						
json	Specifies output in JSON format.						

NX-API request elements are sent to the switch 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 11: NX-API Request Elements**

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

NX-API Request Element	Description
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> <li>• <b>cli_show</b> CLI <b>show</b> commands that expect structured output. If the command does not support XML output, an error message is returned.</li> <li>• <b>cli_show_ascii</b> CLI <b>show</b> commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes.</li> <li>• <b>cli_conf</b> CLI configuration commands.</li> <li>• <b>bash</b> Bash commands. Most non-interactive Bash commands are supported by NX-API.</li> </ul> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Each command is only executable with the current user's authority.</li> <li>• 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.</li> <li>• A maximum of 10 consecutive <b>show</b> commands are supported. If the number of <b>show</b> commands exceeds 10, the 11th and subsequent commands are ignored.</li> <li>• No interactive commands are supported.</li> </ul>

NX-API Request Element	Description						
<i>chunk</i>	<p>Some <b>show</b> 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 <b>show</b> 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><b>Note</b> Only <b>show</b> commands support chunking. When a series of <b>show</b> 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 &lt; or &gt;, are converted to form a valid XML message (&lt; is converted into &amp;lt; &gt; is converted into &amp;gt;).</p> <p>You can use XML SAX to parse the chunked output.</p> <p><b>Note</b> 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>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>						
<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, <b>show</b> commands are cli_show message type and are not supported in cli_conf mode.</p> <p><b>Note</b> Except for <b>bash</b>, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.)</p> <p>For <b>bash</b>, multiple commands are separated with ";". (The ; is <b>not</b> surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="824 1583 1523 1810"> <tr> <td data-bbox="824 1583 948 1656">cli_show</td> <td data-bbox="948 1583 1523 1656">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="824 1656 948 1730">cli_conf</td> <td data-bbox="948 1656 1523 1730">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="824 1730 948 1810">bash</td> <td data-bbox="948 1730 1523 1810">cd /bootflash;mkdir new_dir</td> </tr> </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						

NX-API Request Element	Description	
<i>output_format</i>	The available output message formats are the following:	
	xml	Specifies output in XML format.
	json	Specifies output in JSON format.
	<p><b>Note</b></p> <p>The Cisco Nexus 3500 platform switches 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>	

## NX-API Response Elements

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

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

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

**Table 12: NX-API Response Elements**

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

NX-API Response Element	Description
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution. NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry ( <a href="http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml">http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml</a> ).
msg	Error message associated with the returned error code.

## About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard designed for human-readable data and is an alternative to XML. JSON was originally designed from JavaScript, but it is language-independent data format. The JSON/CLI Execution is currently supported in Cisco Nexus 3500 platform switches.



**Note** The NX-API/JSON functionality is now available on the Cisco Nexus 3500 platform switches.

The two primary Data Structures that are supported in some way by nearly all modern programming languages are as follows:

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

JSON/JSON-RPC/XML output for a show command can also be accessed via sandbox.

## Notes about JSON

The following are notes about JSON:

- The **show run | xml** command and **show run | json** command are not supported (NX-OS 6.0(2)A8(9) and earlier).
- All JSON output is returned as a string (NX-OS 6.0(2)A8(9) and earlier).
- A JSON-RPC integer is always output as an integer and not as string (NX-OS 6.0(2)A8(9) and earlier).

## CLI Execution

### Show\_Command | json

#### Example Code

```
BLR-VXLAN-NPT-CR-179# 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-Disput
e"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}}}
BLR-VXLAN-NPT-CR-179#
```

## XML and JSON Supported Commands

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
- Introduced in NX-OS release 9.3(1), JSON Native and JSON Pretty Native displays JSON output faster and more efficiently by bypassing an extra layer of command interpretation. JSON Native and JSON Pretty Native preserve the data type in the output. They display integers as integers instead of converting them to a string for output.

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, JSON Native, JSON Native 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.

### Examples of 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", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "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": "1
2", "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://w
ww.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 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 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://ww.cisco.com/nxos:1.0:lldp">
 <nf:data>
 <show>
 <lldp>

```

```

<timers>
 <__XML__OPT_Cmd_lldp_show_timers__readonly__>
 <__readonly__>
 <ttd>120</ttd>
 <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)#

```

This example shows how to display ACL statistics in XML format.

```

switch-1(config-acl)# show ip access-lists acl-test1 | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:1.0:aclmgr" xmlns:nf="urn:ietf:p
arams:xml:ns:netconf:base:1.0">
 <nf:data>
 <show>
 <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
 <ip_ipv6_mac>ip</ip_ipv6_mac>
 <access-lists>
 <__XML__OPT_Cmd_show_acl_name>
 <name>acl-test1</name>
 <__XML__OPT_Cmd_show_acl_capture>
 <__XML__OPT_Cmd_show_acl_expanded>
 <__XML__OPT_Cmd_show_acl__readonly__>
 <__readonly__>
 <TABLE_ip_ipv6_mac>
 <ROW_ip_ipv6_mac>
 <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
 <show_summary>0</show_summary>
 <acl_name>acl-test1</acl_name>
 <statistics>enable</statistics>
 <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
 <TABLE_seqno>
 <ROW_seqno>
 <seqno>10</seqno>
 <permitdeny>permit</permitdeny>
 <ip>ip</ip>
 <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
 <dest_any>any</dest_any>
 </ROW_seqno>
 </TABLE_seqno>
 </ROW_ip_ipv6_mac>
 </TABLE_ip_ipv6_mac>
 </__readonly__>
 </__XML__OPT_Cmd_show_acl__readonly__>
 </__XML__OPT_Cmd_show_acl_expanded>
 </__XML__OPT_Cmd_show_acl_capture>
 </__XML__OPT_Cmd_show_acl_name>
 </access-lists>
 </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
</show>
</nf:data>
</nf:rpc-reply>

```



```
]]>]]>
switch-1(config-acl)#
```

This example shows how to display ACL statistics in JSON format.

```
switch-1(config-acl)# show ip access-lists acl-test1 | json
{"TABLE_ip_ipv6_mac": {"ROW_ip_ipv6_mac": {"op_ip_ipv6_mac": "ip", "show_summary": "0", "acl_name": "acl-test1", "statistics": "enable", "frag_opt_permit_deny": "permit-all", "TABLE_seqno": {"ROW_seqno": {"seqno": "10", "permitdeny": "permit", "ip": "ip", "src_ip_prefix": "192.0.2.1/24", "dest_any": "any"}}}}}}
switch-1(config-acl)#
```

This example shows how to display the switch's redundancy information in JSON Pretty Native format.

```
switch-1# show system redundancy status | json-pretty native
{
 "rdn_mode_admin": "HA",
 "rdn_mode_oper": "None",
 "this_sup": "(sup-1)",
 "this_sup_rdn_state": "Active, SC not present",
 "this_sup_sup_state": "Active",
 "this_sup_internal_state": "Active with no standby",
 "other_sup": "(sup-1)",
 "other_sup_rdn_state": "Not present"
}
switch-1#
```

The following example shows how to display the switch's redundancy status in JSON format.

```
switch-1# show system redundancy status | json
{"rdn_mode_admin": "HA", "rdn_mode_oper": "None", "this_sup": "(sup-1)", "this_sup_rdn_state": "Active, SC not present", "this_sup_sup_state": "Active", "this_sup_internal_state": "Active with no standby", "other_sup": "(sup-1)", "other_sup_rdn_state": "Not present"}
nxosv2#
switch-1#
```

The following example shows how to display the IP route summary in XML format.

```
switch-1# show ip route summary | xml
<?xml version="1.0" encoding="ISO-8859-1"?> <nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:urib" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
 <nf:data>
 <show>
 <ip>
 <route>
 <_XML_OPT_Cmd_urib_show_ip_route_command_ip>
 <_XML_OPT_Cmd_urib_show_ip_route_command_unicast>
 <_XML_OPT_Cmd_urib_show_ip_route_command_topology>
 <_XML_OPT_Cmd_urib_show_ip_route_command_l3vm-info>
 <_XML_OPT_Cmd_urib_show_ip_route_command_rpf>
 <_XML_OPT_Cmd_urib_show_ip_route_command_ip-addr>
 <_XML_OPT_Cmd_urib_show_ip_route_command_protocol>
 <_XML_OPT_Cmd_urib_show_ip_route_command_summary>
 <_XML_OPT_Cmd_urib_show_ip_route_command_vrf>
 <_XML_OPT_Cmd_urib_show_ip_route_command__readonly__>
 <_readonly__>
 <TABLE_vrf>
 <ROW_vrf>
 <vrf-name-out>default</vrf-name-out>
 <TABLE_addrf>
 <ROW_addrf>
 <addrf>ipv4</addrf>
 <TABLE_summary>
 <ROW_summary>
```

```

<routes>938</routes>
<paths>1453</paths>
<TABLE_unicast>
 <ROW_unicast>
 <clientnameuni>am</clientnameuni>
 <best-paths>2</best-paths>
 </ROW_unicast>
 <ROW_unicast>
 <clientnameuni>local</clientnameuni>
 <best-paths>105</best-paths>
 </ROW_unicast>
 <ROW_unicast>
 <clientnameuni>direct</clientnameuni>
 <best-paths>105</best-paths>
 </ROW_unicast>
 <ROW_unicast>
 <clientnameuni>broadcast</clientnameuni>
 <best-paths>203</best-paths>
 </ROW_unicast>
 <ROW_unicast>
 <clientnameuni>ospf-10</clientnameuni>
 <best-paths>1038</best-paths>
 </ROW_unicast>
</TABLE_unicast>
<TABLE_route_count>
 <ROW_route_count>
 <mask_len>8</mask_len>
 <count>1</count>
 </ROW_route_count>
 <ROW_route_count>
 <mask_len>24</mask_len>
 <count>600</count>
 </ROW_route_count>
 <ROW_route_count>
 <mask_len>31</mask_len>
 <count>13</count>
 </ROW_route_count>
 <ROW_route_count>
 <mask_len>32</mask_len>
 <count>324</count>
 </ROW_route_count>
</TABLE_route_count>
</ROW_summary>
</TABLE_summary>
</ROW_addrf>
</TABLE_addrf>
</ROW_vrf>
</TABLE_vrf>
</__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
</__XML__OPT_Cmd_urib_show_ip_route_command_summary>
</__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
</__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
</__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
</__XML__OPT_Cmd_urib_show_ip_route_command_topology>
</__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip>
</route>
</ip>
</show>
</nf:data>
</nf:rpc-reply>

```

```
]]>]]>
switch-1#
```

The following example shows how to display the switch's OSPF routing parameters in JSON Native format.

```
switch-1# show ip ospf | json native
{"TABLE_ctx":{"ROW_ctx":[{"ptag":"Blah","instance_number":4,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","instance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_number":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"}]}]}
switch-1#
```

The following example shows how to display OSPF routing parameters in JSON Pretty Native format.

```
switch-1# show ip ospf | json-pretty native
{
 "TABLE_ctx": {
 "ROW_ctx": [{
 "ptag": "Blah",
 "instance_number": 4,
 "cname": "default",
 "rid": "0.0.0.0",
 "stateful_ha": "true",
 "gr_ha": "true",
 "gr_planned_only": "true",
 "gr_grace_period": "PT60S",
```

```

 "gr_state": "inactive",
 "gr_last_status": "None",
 "support_tos0_only": "true",
 "support_opaque_lsa": "true",
 "is_abr": "false",
 "is_asbr": "false",
 "admin_dist": 110,
 "ref_bw": 40000,
 "spf_start_time": "PT0S",
 "spf_hold_time": "PT1S",
 "spf_max_time": "PT5S",
 "lsa_start_time": "PT0S",
 "lsa_hold_time": "PT5S",
 "lsa_max_time": "PT5S",
 "min_lsa_arr_time": "PT1S",
 "lsa_aging_pace": 10,
 "spf_max_paths": 8,
 "max_metric_adver": "false",
 "asext_lsa_cnt": 0,
 "asext_lsa_crc": "0",
 "asopaque_lsa_cnt": 0,
 "asopaque_lsa_crc": "0",
 "area_total": 0,
 "area_normal": 0,
 "area_stub": 0,
 "area_nssa": 0,
 "act_area_total": 0,
 "act_area_normal": 0,
 "act_area_stub": 0,
 "act_area_nssa": 0,
 "no_discard_rt_ext": "false",
 "no_discard_rt_int": "false"
 }, {
 "ptag": "100",
 "instance_number": 3,
 "cname": "default",
 "rid": "0.0.0.0",
 "stateful_ha": "true",
 "gr_ha": "true",
 "gr_planned_only": "true",
 "gr_grace_period": "PT60S",
 "gr_state": "inactive",

 ... content deleted for brevity ...

 "max_metric_adver": "false",
 "asext_lsa_cnt": 0,
 "asext_lsa_crc": "0",
 "asopaque_lsa_cnt": 0,
 "asopaque_lsa_crc": "0",
 "area_total": 0,
 "area_normal": 0,
 "area_stub": 0,
 "area_nssa": 0,
 "act_area_total": 0,
 "act_area_normal": 0,
 "act_area_stub": 0,
 "act_area_nssa": 0,
 "no_discard_rt_ext": "false",
 "no_discard_rt_int": "false"
 }
}
switch-1#

```

The following example shows how to display the IP route summary in JSON format.

```
switch-1# show ip route summary | json
{"TABLE_vrf": {"ROW_vrf": {"vrf-name-out": "default", "TABLE_addrf": {"ROW_addrf": {"addrf": "ipv4", "TABLE_summary": {"ROW_summary": {"routes": "938", "paths": "1453", "TABLE_unicast": {"ROW_unicast": [{"clientnameuni": "am", "best-paths": "2"}, {"clientnameuni": "local", "best-paths": "105"}, {"clientnameuni": "direct", "best-paths": "105"}, {"clientnameuni": "broadcast", "best-paths": "203"}, {"clientnameuni": "ospf-10", "best-paths": "1038"}]}}, "TABLE_route_count": {"ROW_route_count": [{"mask_len": "8", "count": "1"}, {"mask_len": "24", "count": "600"}, {"mask_len": "31", "count": "13"}, {"mask_len": "32", "count": "324"}]}}}}}}}}}
switch-1#
```

The following example shows how to display the IP route summary in JSON Pretty format.

```
switch-1# show ip route summary | json-pretty
{
 "TABLE_vrf": {
 "ROW_vrf": {
 "vrf-name-out": "default",
 "TABLE_addrf": {
 "ROW_addrf": {
 "addrf": "ipv4",
 "TABLE_summary": {
 "ROW_summary": {
 "routes": "938",
 "paths": "1453",
 "TABLE_unicast": {
 "ROW_unicast": [
 {
 "clientnameuni": "am",
 "best-paths": "2"
 },
 {
 "clientnameuni": "local",
 "best-paths": "105"
 },
 {
 "clientnameuni": "direct",
 "best-paths": "105"
 },
 {
 "clientnameuni": "broadcast",
 "best-paths": "203"
 },
 {
 "clientnameuni": "ospf-10",
 "best-paths": "1038"
 }
]
 }
 }
 }
 }
 }
 }
 },
 "TABLE_route_count": {
 "ROW_route_count": [
 {
 "mask_len": "8",
 "count": "1"
 },
 {
 "mask_len": "24",
 "count": "600"
 },
 {
 "mask_len": "31",
 "count": "13"
 }
]
 }
}
```



- show lldp traffic interface ethernet x/x
- show process memory
- show process cpu & show process
- show routing vrf all
- show system internal forwarding route summary
- show system resources



**Note** The maximum data size supported by NX-API is 20MB. The following commands are not supported when the 20MB limit has been exceeded (NX-OS 6.0(2)A8(9) and earlier):

- show hardware profile buffer monitor detail
- show hardware profile buffer monitor multicast 1 detail
- show hardware profile buffer monitor multicast 2 detail
- show hardware profile buffer monitor multicast 3 detail

## Examples of 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", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "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": "1
2", "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://w
ww.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 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 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: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>
 </__readonly__>
 </__XML__OPT_Cmd_lldp_show_timers__readonly__>
 </timers>
 </lldp>
 </show>
 </nf:data>
</nf:rpc-reply>

```



```
 <notification_interval>5</notification_interval>
 </__readonly__>
 </__XML__OPT_Cmd_lddp_show_timers__readonly__>
 </timers>
</lldp>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```





# CHAPTER 15

## NX-API Response Codes

- [Table of NX-API Response Codes, on page 161](#)

### Table of NX-API Response Codes

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

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

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



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

**Table 13: 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.
EOC_NOT_ALLOWED_ERR	400	The <code>EOC</code> value is not allowed as session Id in the request.
IN_MSG_ERR	400	Request message is invalid.
MSG_VER_MISMATCH	400	Message version mismatch.

NO_INPUT_CMD_ERR	400	No input command.
SID_NOT_ALLOWED_ERR	400	Invalid character that is entered as a session ID.
PERM_DENY_ERR	401	Permission denied.
CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow <b>show</b> .
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.
RESP_SIZE_LARGE_ERR	413	Response size stopped processing because it exceeded the maximum message size. The maximum is 200 MB.
EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR	429	Maximum number of concurrent chunk requests is exceeded. The maximum is 2.
OBJ_NOT_EXIST	432	Requested object does not exist.
BACKEND_ERR	500	Backend processing error.
CREATE_CHECKPOINT_ERR	500	Error creating a checkpoint.
DELETE_CHECKPOINT_ERR	500	Error deleting a checkpoint.
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.
ROLLBACK_ERR	500	Error executing a rollback.
SERVER_BUSY_ERR	500	Request is rejected because the server is busy.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
VOLATILE_FULL	500	Volatile memory is full. Free up memory space and retry.
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.

CHUNK_ONLY_ALLOWED_IN_SHOW_ERR	501	Response chunking allowed only in <code>show</code> commands.
CHUNK_TIMEOUT	501	Timeout while generating chunk response.
CLI_CMD_NOT_SUPPORTEDED_ERR	501	CLI command not supported.
JSON_NOT_SUPPORTEDED_ERR	501	JSON not supported due to large amount of output.
MALFORMED_XML	501	Malformed XML output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
OUTPUT_REDIRECT_NOT_SUPPORTEDED_ERR	501	Output redirection is not supported.
PIPE_OUTPUT_NOT_SUPPORTEDED_ERR	501	Pipe operation is not supported.
PIPE_OUTPUT_NOT_SUPPORTEDED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
PIPE_NOT_ALLOWED_IN_INPUT	501	Pipe is not allowed for this input type.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
RESP_BIG_USE_CHUNK_ERR	501	Response is greater than the allowed maximum. The maximum is 10 MB. Use XML or JSON output with chunking enabled.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTEDED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.





# CHAPTER 16

## NX-API Developer Sandbox

- NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2), on page 165
- NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later, on page 170

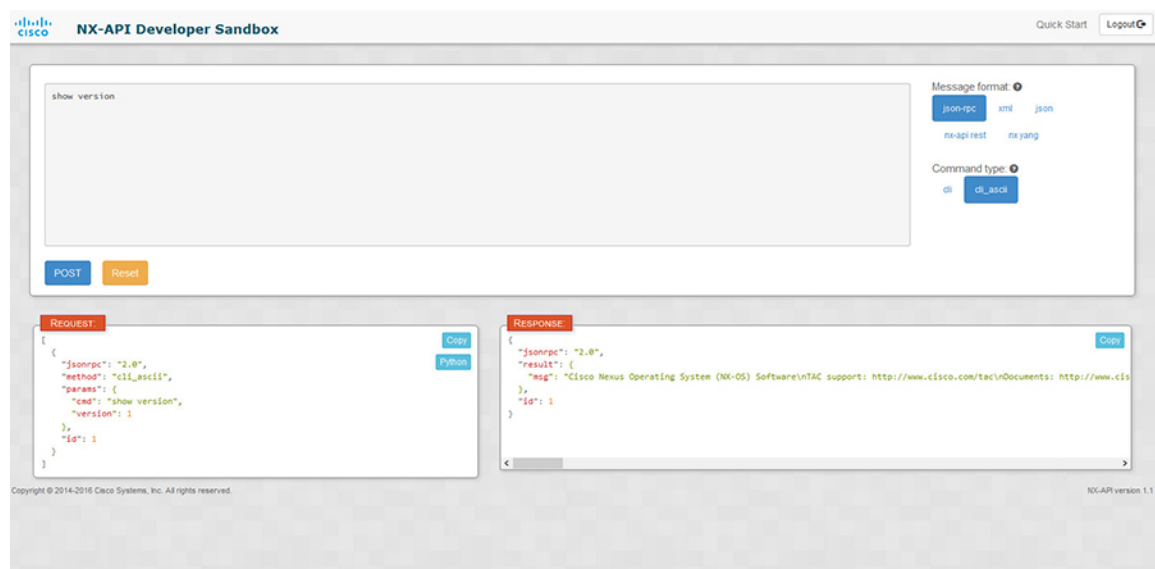
### NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2)

#### 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** Message format and the **model** Command 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 **POST** 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.

## Configuring the Message Format and Command Type

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

*Table 14: NX-OS API Protocols*

Protocol	Description
json-rpc	A standard lightweight remote procedure call (RPC) protocol that can be used to deliver NX-OS CLI commands in a JSON payload. The JSON-RPC 2.0 specification is outlined by <a href="http://jsonrpc.org">jsonrpc.org</a> .
xml	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML payload.
json	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in a JSON payload.
nx-api rest	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, see the <a href="#">Cisco Nexus NX-API References</a> .
nx yang	The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.

When the **Message Format** has been chosen, a set of **Command Type** options are presented just below the **Message Format** control. The **Command Type** setting can constrain the input CLI and can determine the **Request** and **Response** format. The options vary depending on the **Message Format** selection. For each **Message Format**, the following table describes the **Command Type** options:



Table 15: Command Types

Message format	Command type
json-rpc	<ul style="list-style-type: none"> <li>cli — show or configuration commands</li> <li>cli-ascii — show or configuration commands, output without formatting</li> </ul>
xml	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
json	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
nx-api rest	<ul style="list-style-type: none"> <li>cli — configuration commands</li> <li>model — DN and corresponding payload.</li> </ul>
nx yang	<ul style="list-style-type: none"> <li>json — JSON structure is used for payload</li> <li>xml — XML structure is used for payload</li> </ul>

### 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

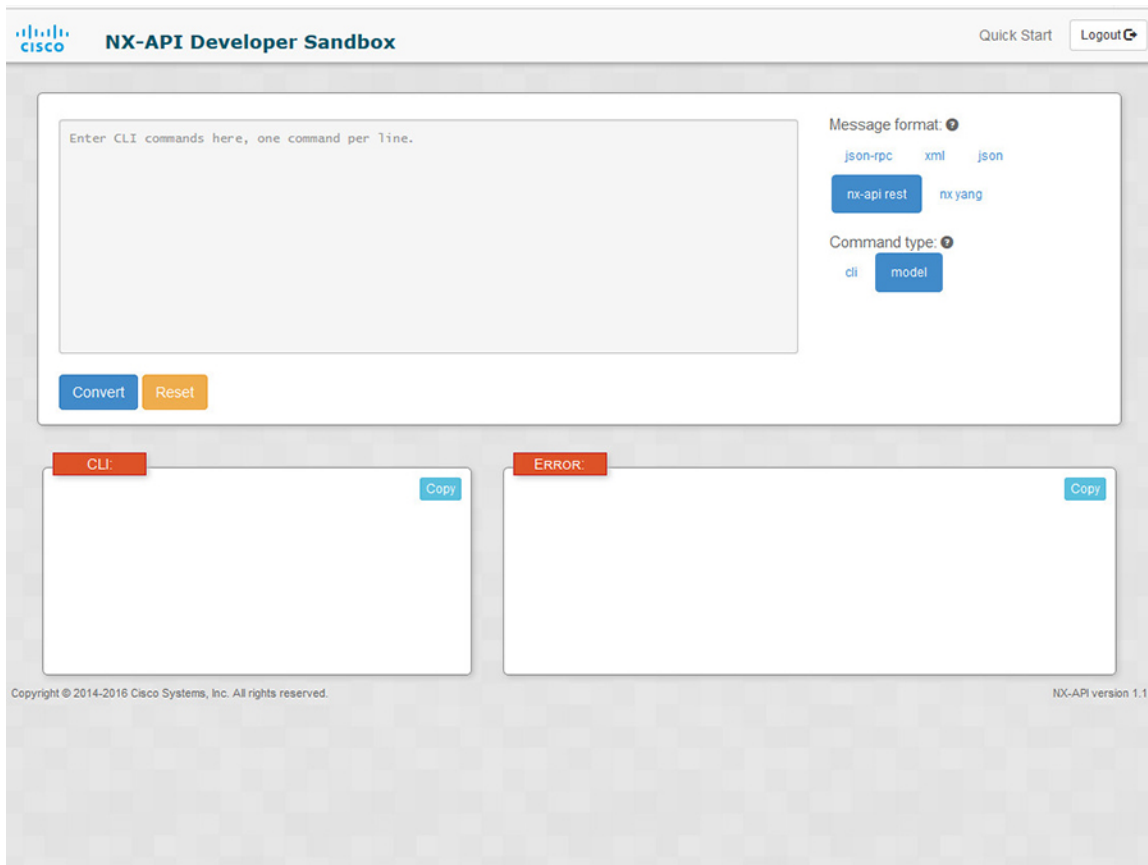
### Using the Developer Sandbox to Convert CLI Commands to Payloads



**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 NX-API CLI chapter. Only configuration commands are supported.

#### Procedure

- Step 1** Configure the **Message Format** and **Command Type** for the API protocol you want to use. For detailed instructions, see [Configuring the Message Format and Command Type, on page 166](#).
- 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.

The screenshot displays the NX-API Developer Sandbox interface. At the top, the Cisco logo and "NX-API Developer Sandbox" title are visible, along with "Quick Start" and "Logout" links. The main area is divided into several sections:

- Input Area:** A text area containing a JSON payload: 

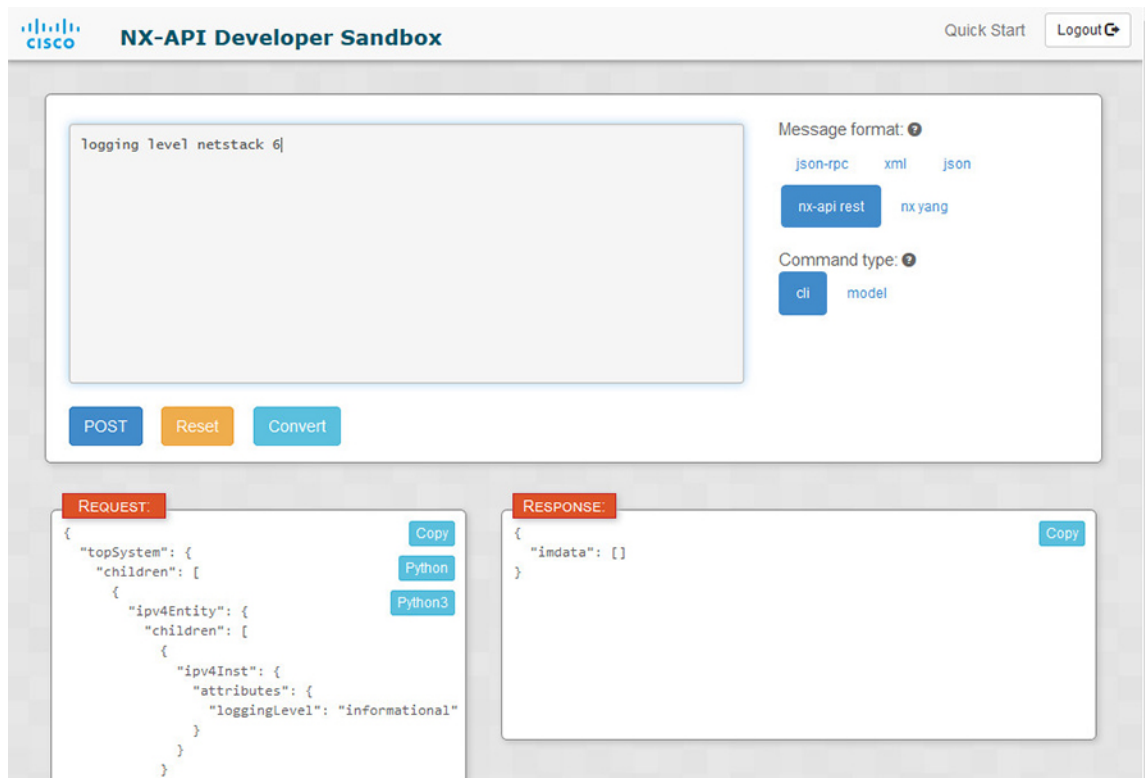
```
api/mo/sys.json
{
 "topSystem": {
 "attributes": {
 "name": "REST2CLI"
 }
 }
}
```
- Message format:** Radio buttons for "json-rpc", "xml", and "json". The "json" option is selected.
- Command type:** Radio buttons for "cli" and "model". The "model" option is selected.
- Buttons:** "Convert" (blue) and "Reset" (orange) buttons are located below the input area.
- CLI Pane:** A box with a red header "CLI:" containing the text "hostname REST2CLI" and a "Copy" button.
- ERROR Pane:** A box with a red header "ERROR:" and a "Copy" button, currently empty.
- Footer:** Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved. and NX-API version 1.1.
- Terminal:** A terminal window at the bottom shows "Waiting for bam.nr-data.net..."

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

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

**Warning**

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



**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 obtain a Python implementation of the request on the clipboard by clicking **Python** in the **Request** pane.

## NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later

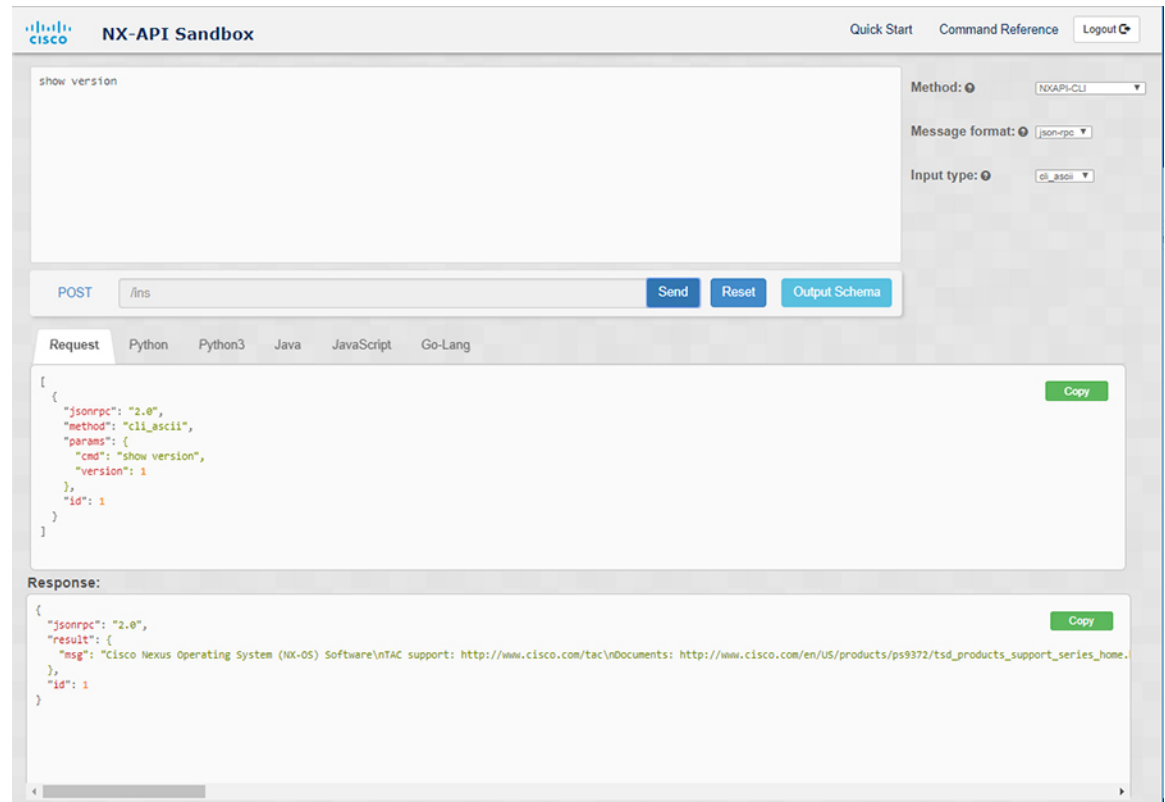
### About the NX-API Developer Sandbox

The Cisco 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 (middle pane), and Response (bottom pane) — as shown in the figure below. The designated name (DN) field is located between the Command and Request panes (seen in the figure below located between the **POST** and **Send** options).

The Request pane also has a series of tabs. Each tab represents a different language: **Python**, **Python3**, **Java**, **JavaScript**, and **Go-Lang**. Each tab enables you to view the request in the respective language. For example, after converting CLI commands into an XML or JSON payload, click the **Python** tab to view the request in Python, which you can use to create scripts.

Figure 2: NX-API Developer Sandbox with Example Request and Output Response



Controls in the Command pane enable you to choose a supported API, such as NX-API REST, an input type, such as model (payload) or CLI, and a message format, such as XML or JSON. The available options vary depending on the chosen method.

When you choose the NXAPI-REST (DME) method, type or paste one or more CLI commands into the Command pane, and click **Convert**, the web form converts the commands into a REST 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 (by choosing the **POST** option and clicking **SEND**), the Response pane displays the API response. For more information, see [Using the Developer Sandbox to Convert CLI Commands to REST Payloads, on page 177](#)

Conversely, the Cisco NX-API Developer Sandbox checks the payload for configuration errors then displays the equivalent CLIs in the Response pane. For more information, see [Using the Developer Sandbox to Convert from REST Payloads to CLI Commands, on page 179](#)

## 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. For example, configuring a BGP router requires first enabling BGP with the **feature bgp** command. Similarly, configuring an OSPF router requires first enabling OSPF with the **feature ospf** command.

This also applies to **evpn esi multihoming**, which enables its dependent commands such as **evpn multihoming core-tracking**. For more information about enabling features to access feature dependent commands, see the [Cisco Nexus 9000 Configuration Guides](#) and [Cisco Nexus 3000 Configuration Guides](#).

- 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.
- CLI to model or xml conversion will not happen for OSPFv2 interface commands until you explicitly enable OSPF on interface by configuring router instance and area using **[no] ip router ospf <tag> area {<area-id-ip> | <area-id-int>} [secondaries none]** command.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- When you use XML or JSON as the Message Type for CLI input, you can use semicolon to separate multiple commands on the same line. However, when you use JSON RPC as the Message Type for CLI input, you cannot enter multiple commands on the same line and separate them with a semicolon (;).

For example, assume that you want to send **show hostname** and **show clock** commands through JSON RPC as the following.

In the Sandbox, you enter the CLIs as follows.

```
show hostname ; show clock
```

In the JSON RPC request, the input is formatted as follows.

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

When you send the request, the response returns the following error.

```
{
 "jsonrpc": "2.0",
 "error": {
 "code": -32602,
 "message": "Invalid params",
 "data": {
 "msg": "Request contains invalid special characters"
 }
 },
 "id": 1
}
```

This situation occurs because the Sandbox parses each command in a JSON RPC request as individual items and assigns an ID to each. When using JSON RPC requests, you cannot use internal punctuation to separate multiple commands on the same line. Instead, enter each command on a separate line and the request completes successfully.

Continuing with the same example, enter the commands as follows in the NX-API CLI.

```
show hostname
show clock
```

In the request, the input is formatted as follows.

```
[
 {
 "jsonrpc": "2.0",
 "method": "cli",
 "params": {
 "cmd": "show hostname",
 "version": 1
 },
 "id": 1
 },
 {
 "jsonrpc": "2.0",
 "method": "cli",
 "params": {
 "cmd": "show clock",
 "version": 1
 },
 "id": 2
 }
]
```

The response completes successfully.

```
[
 {
 "jsonrpc": "2.0",
 "result": {
 "body": {
 "hostname": "switch-1"
 }
 },
 "id": 1
 },
 {
 "jsonrpc": "2.0",
 "result": {
 "body": {
 "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
 "time_source": "NTP"
 }
 },
 "id": 2
 }
]
```

## 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 Cisco NX-API Developer Sandbox supports the following API protocols:

**Table 16: 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)	<p>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. The NXAPI-REST (DME) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"> <li>• <b>POST</b></li> <li>• <b>GET</b></li> <li>• <b>PUT</b></li> <li>• <b>DELETE</b></li> </ul> <p>For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see <a href="https://developer.cisco.com/site/cisco-nexus-nx-api-references/">https://developer.cisco.com/site/cisco-nexus-nx-api-references/</a>.</p>
RESTCONF (Yang)	<p>The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.</p> <p>The RESTCONF (Yang) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"> <li>• <b>POST</b></li> <li>• <b>GET</b></li> <li>• <b>PUT</b></li> <li>• <b>PATCH</b></li> <li>• <b>DELETE</b></li> </ul>

When you choose the **Method**, a set of **Message format** or **Input type** options are displayed in a drop-down list. The **Message format** can constrain the input CLI and determine the **Request** and **Response** format. The options vary depending on the **Method** you choose.

The following table describes the **Input/Command type** options for each **Message format**:

**Table 17: Command Types**

Method	Message format	Input/Command type
NXAPI-CLI	json-rpc	<ul style="list-style-type: none"> <li>• cli — show or configuration commands</li> <li>• cli-ascii — show or configuration commands, output without formatting</li> <li>• cli-array — show commands. Similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [ ].</li> </ul>



Method	Message format	Input/Command type
NXAPI-CLI	xml	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
NXAPI-CLI	json	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> </ul> <p><b>Note</b> Beginning with Cisco NX-OS Release 9.3(3), the cli_show_array command is recommended over the cli_show command.</p> <ul style="list-style-type: none"> <li>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 [ ].</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
NXAPI-REST (DME)		<ul style="list-style-type: none"> <li>cli — CLI to model conversion</li> <li>model — Model to CLI conversion.</li> </ul>
RESTCONF (Yang)	<ul style="list-style-type: none"> <li>json — JSON structure is used for payload</li> <li>xml — XML structure is used for payload</li> </ul>	

### Output Chunking

JSON and XML NX-API message formats enable you to receive large show command responses in 10-MB chunks. When received, the chunks are concatenated to create a valid JSON object or XML structure. To view a sample script that demonstrates output chunking, click the following link and choose the directory that corresponds to Release 9.3x: [Cisco NX-OS NXAPI](#).




---

**Note** For chunk JSON mode, the browser or python script part does not provide the valid JSON output (there will be no closing tags). To use chunk mode and get valid JSON, use the script provided in the directory.

---

You receive the first chunk in the immediate command response, which also includes a **sid** field that contains a session Id. To retrieve the next chunk, you enter the session Id from the previous chunk in the **SID** text box. You repeat the process until reaching the last response, which is indicated by the **eof** (end of content) value in the **sid** field.

Chunk mode is available when using the **NXAPI-CLI** method with the **JSON** or **XML** format type and the **cli\_show**, **cli\_show\_array**, or **cli\_show\_ascii** command type. For more information about configuring the chunk mode, see the *Chunk Mode Fields* table.




---

**Note** NX-API supports a maximum of 2 chunking sessions.

---

**Table 18: Chunk Mode Fields**

Field Name	Description
<b>Enable Chunk Mode</b>	Click to place a check mark in the <b>Enable Chunk Mode</b> check box to enable chunking. When you enable chunk mode, responses that exceed 10 MB are sent in multiple chunks of up to 10 MB in size.
<b>SID</b>	Enter the session Id of the previous response in the <b>SID</b> text box to retrieve the next chunk of the response message.  <b>Note</b> Only alphanumeric characters and ‘_’ are allowed. Invalid characters receive an error.

## Using the Developer Sandbox

You can use the Cisco NX-API Developer Sandbox to make multiple conversions, including the following:

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

**Tip**

- Online help is available by clicking the help icons (?) next to the field names located in the upper-right corner of the Cisco NX-API Developer Sandbox window.
- For additional details, such as response codes and security methods, see the *NX-API CLI* chapter.
- Only configuration commands are supported.

The Cisco NX-API Developer Sandbox enables you to convert CLI commands to REST payloads.

### Procedure

**Step 1** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.

The **Input** type drop-down list appears.

**Step 2** Click the **Input** type drop-down list and choose **cli**.

**Step 3** 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.

The screenshot shows the Cisco NX-API Developer 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 has a large text input field with the placeholder "Enter DME payload here.". To the right of this field are two dropdown menus: "Method" (set to "NXAPI-REST (DME)") and "Input type" (set to "model"). Below the input field is a smaller text box containing the text "/api/mo/sys.json". To the right of this box are three buttons: "Send" (blue), "Reset" (orange), and "Convert" (blue). The bottom pane is divided into two sections: "Request" and "Response". The "Request" section has a "Copy" button (green) and is currently empty. The "Response" section also has a "Copy" button (green) and is currently empty. Below the "Request" and "Response" sections are tabs for different languages: "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang".

**Step 4** Click **Convert**.

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.

**Step 5** (Optional) To send a valid payload as an API call to the switch, click **Send**.

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.

The screenshot shows the NX-API Sandbox interface. At the top, there is a header with the Cisco logo and the text "NX-API Sandbox". To the right of the header are links for "Quick Start", "DME Documentation", "Model Browser", and a "Logout" button. Below the header is a text area containing the CLI command "logging level ~~notstack~~ 6". To the right of this text area are two dropdown menus: "Method" set to "NX-API-REST (DME)" and "Input type" set to "cli". Below these is a form with a "POST" dropdown, a text input field containing "/api/mo/sys.json", and three buttons: "Send", "Reset", and "Convert". Below the form is a "Request" pane with tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" pane shows a JSON payload:
 

```
{
 "topsystem": {
 "children": [
 {
 "ipv4Entity": {
 "children": [
 {
 "ipv4Inst": {
 "attributes": {
 "logginglevel": "informational"
 }
 }
 }
]
 }
 }
]
 }
}
```

 To the right of the JSON payload is a green "Copy" button. Below the "Request" pane is a "Response" pane showing a JSON response:
 

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

 To the right of the JSON response is another green "Copy" button.

**Step 6** (Optional) To obtain the DN for an MO in the payload:

- a. From the **Request** pane, choose **POST**.
- b. Click the **Convert** drop-down list and choose **Convert (with DN)**.

The payload appears with with a **dn** field that contains the DN that corresponds to each MO in the payload.

**Step 7** (Optional) To overwrite the current configuration with a new configuration:

- a. Click the **Convert** drop-down list and choose **Convert (for Replace)**. The **Request** pane displays a payload with a **status** field set to **replace**.
- b. From the **Request** pane, choose **POST**.
- c. Click **Send**.

The current configuration is replaced with the posted configuration. For example, if you start with the following configuration:

```
interface eth1/2
 description test
 mtu 1501
```

Then use **Convert (for Replace)** to POST the following configuration:

```
interface eth1/2
 description testForcr
```

The `mtu` configuration is removed and only the new description (`testForcr`) is present under the interface. This change is confirmed when entering **show running-config**.

- Step 8** (Optional) To copy the contents of a pane, such as the **Request** or **Response** pane, click **Copy**. The contents of the respective pane is copied to the clipboard.
- Step 9** (Optional) To convert the request into an of the formats listed below, click 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

The Cisco NX-API Developer Sandbox enables you to convert REST payloads to corresponding CLI commands. This option is only available for the NXAPI-REST (DME) method.



### Tip

- Online help is available by clicking help icons (?) next to the Cisco NX-API Developer Sandbox field names. Click a help icon get information about the respective field.
- For additional details, such as response codes and security methods, see the chapter *NX-API CLI*.
- The top-right corner of the Cisco NX-API Developer Sandbox contains links for additional information. The links that appear depend on the **Method** you choose. The links that appear for the NXAPI-REST (DME) method:
    - **NX-API References**—Enables you to access additional NX-API documentation.
    - **DME Documentation**—Enables you to access the NX-API DME Model Reference page.
    - **Model Browser**—Enables you 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** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.

### Example:

The screenshot shows the NX-API Sandbox interface. At the top, there is a navigation bar with the Cisco logo, the title "NX-API Sandbox", and links for "Quick Start", "DME Documentation", "Model Browser", and "Logout". Below the navigation bar is a large text area for entering the DME payload. To the right of this area are two dropdown menus: "Method" set to "NXAPI-REST (DME)" and "Input type" set to "model". Below these is a text input field containing the path "/api/mo/sys.json" and three buttons: "Send", "Reset", and "Convert". Underneath the input field are tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing a large empty text area with a "Copy" button in the top right corner. Below the Request pane is a "Response:" section, also with a large empty text area and a "Copy" button in the top right corner.

**Step 2** Click the **Input Type** drop-down list and choose **model**.

**Step 3** Enter the designated name (DN) that corresponds to the payload in the field above the Request pane.

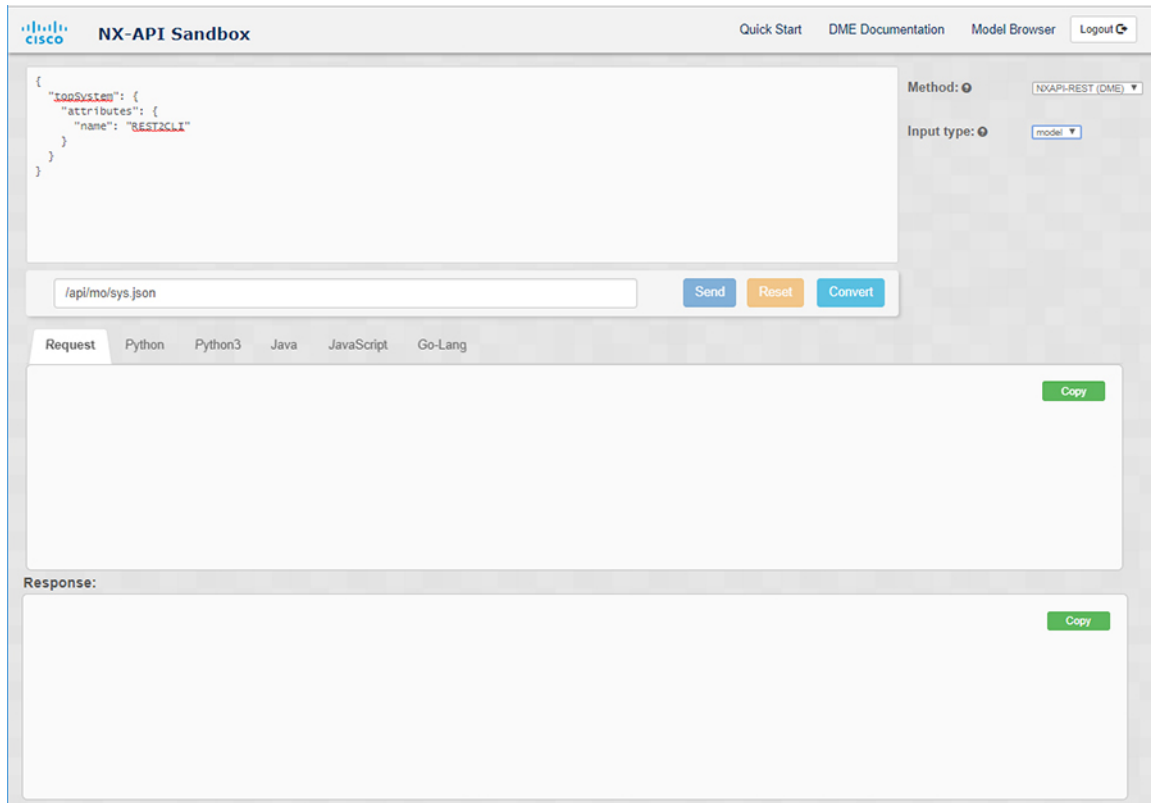
**Step 4** Enter the payload in the Command pane.

**Step 5** Click **Convert**.

### Example:

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

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



The screenshot displays the NX-API Developer Sandbox interface. At the top, there is a navigation bar with the Cisco logo, the title "NX-API Sandbox", and links for "Quick Start", "DME Documentation", "Model Browser", and "Logout". The main area is divided into several sections:

- Request Payload:** A text area containing a JSON payload: 

```
{ "sysSystem": { "attributes": { "name": "REST2CLI" } } }
```
- Method:** A dropdown menu set to "NXAPI-REST (DME)".
- Input type:** A dropdown menu set to "model".
- URL:** A text input field containing "/api/mo/sys.json".
- Buttons:** "Send" (blue), "Reset" (orange), and "Convert" (blue).
- Request Tab:** A tab labeled "Request" with sub-tabs for "Python", "Python3", "Java", "JavaScript", and "Go-Lang". Below this is a large empty text area with a "Copy" button.
- Response:** A section labeled "Response:" with a large empty text area and a "Copy" button.

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 Developer Sandbox interface. At the top, there is a navigation bar with the Cisco logo, the title "NX-API Sandbox", and links for "Quick Start", "DME Documentation", "Model Browser", and "Logout". The main area is divided into two sections. The top section contains a text area with a JSON payload: 

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

. To the right of this text 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 bottom section is titled "Request" and has tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing the converted CLI command "hostname REST2CLI" with a "Copy" button. Below this is a "Response:" section, which is currently empty, also featuring a "Copy" button.

**Note**

The Cisco NX-API 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 19: 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 <b>Error</b> pane will return an error related to the attribute.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> 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 <b>Error</b> Pane will return an error related to the unsupported MO.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> 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 the help icon (?) in the upper-right corner of the Cisco NX-API Developer 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** Click the **Method** drop-down list and choose **RESTCONF (Yang)**.

### Example:

The screenshot shows the Cisco NX-API Sandbox interface. At the top, there are navigation links for "Quick Start", "Yang Documentation", "Yang Models", and "Logout". The main area contains a text input field with the text "Logging level netstack". To the right of this field, there are two dropdown menus: "Method:" set to "RESTCONF (Yang)" and "Message format:" set to "json". Below these fields is a "POST" dropdown menu and a text input field containing the URL "restconf/data/Cisco-NX-OS-device:System/". There are three buttons: "Send" (orange), "Reset" (blue), and "Convert" (light blue). Below the URL field, there are tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing a large empty text area with a "Copy" button. Below the "Request" area is a "Response:" section, also with a large empty text area and a "Copy" button.

**Step 2** Click **Message format** and choose either **json** or **xml**.

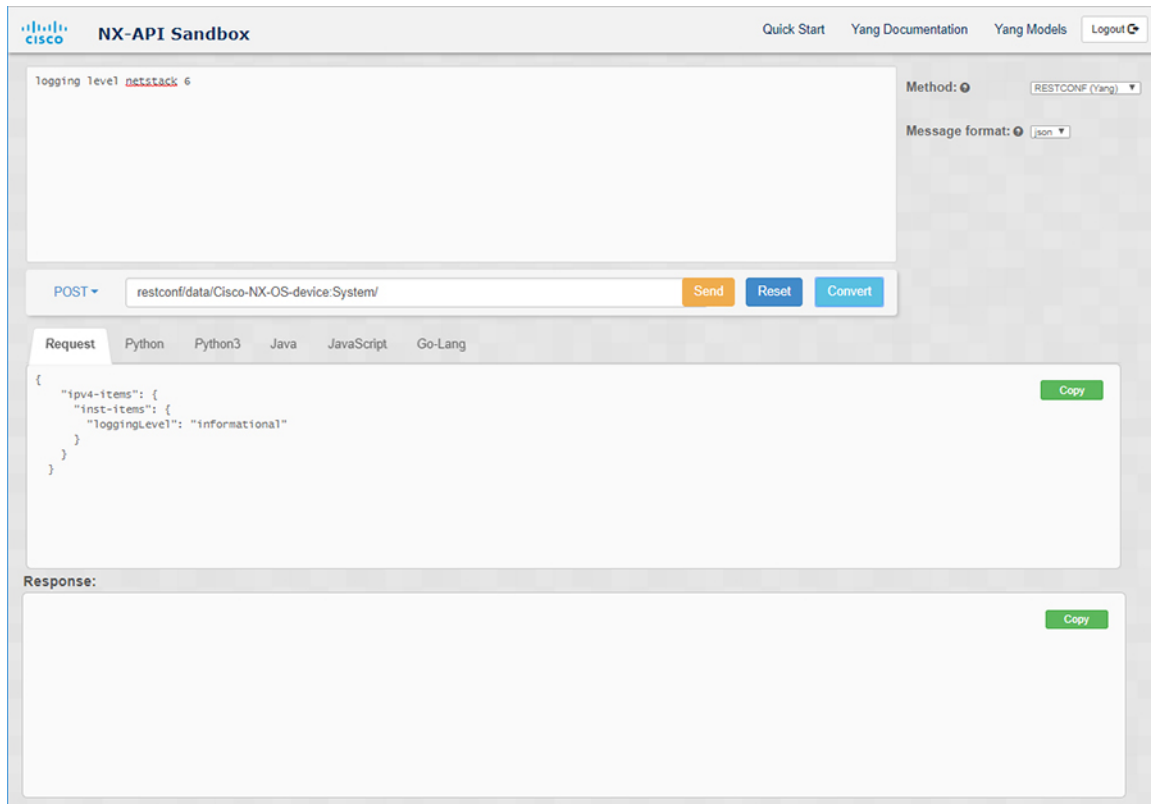
**Step 3** Enter a command in the text entry box in the top pane.

**Step 4** Choose a message format.

**Step 5** Click **Convert**.

**Example:**

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



The screenshot shows the NX-API Developer Sandbox interface. At the top, there is a header with the Cisco logo and the text "NX-API Sandbox". To the right of the header are links for "Quick Start", "Yang Documentation", "Yang Models", and a "Logout" button. Below the header, there is a text area containing the command `logging level netstack 6`. To the right of this text area are two dropdown menus: "Method" set to "RESTCONF (Yang)" and "Message format" set to "json". Below these is a "POST" dropdown menu and a text input field containing the URL `restconf/data/Cisco-NX-OS-device:System/`. To the right of the input field are three buttons: "Send", "Reset", and "Convert". Below the input field is a tabbed interface with tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing a JSON request body: 

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

 To the right of the JSON is a green "Copy" button. Below the request is a "Response:" label and an empty text area with a green "Copy" button to its right.

**Example:**

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

**Note**

When converting a negated CLI to a Yang payload using the XML or JSON message format, the sandbox throws a warning and disables the **Send** option. The warning message that appears depends on the message format:

- For the XML message format — "This is a Netconf payload as it is being generated for DELETE operation(s), hence SEND option is disabled for Restconf!"
- For the JSON message format—"This is a gRPC payload as it is being generated for DELETE operation(s), hence SEND option is disabled for Restconf!"

**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

**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.



## CHAPTER 17

# XML Support for ABM and LM in N3500

- [XML Support for ABM and LM in N3500](#) , on page 187

## XML Support for ABM and LM in N3500

The following commands show XML Output for ABM and LM:

### **show hardware profile buffer monitor sampling**

**CLI :**

```
MTC-8(config)# show hardware profile buffer monitor sampling
```

```
Sampling CLI issued at: 05/25/2016 04:18:56
```

```
Sampling interval: 200
```

**XML :**

```
MTC-8(config)# show hardware profile buffer monitor sampling | 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:mtc_usd_cli">
```

```
<nf:data>
```

```
<show>
```

```
<hardware>
```

```
<profile>
```

```
<buffer>
```

```
<monitor>
```

```
<__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
```

```
<__XML__OPT_Cmd_show_hardware_profile_buffer_monitor__readonly__>
```

```
<__readonly__>
```

```
<cmd_name>Sampling CLI</cmd_name>
```

```

<cmd_issue_time>05/25/2016 04:19:12</cmd_issue_time>

<TABLE_sampling>

 <ROW_sampling>

 <sampling_interval>200</sampling_interval>

 </ROW_sampling>

</TABLE_sampling>

</__readonly__>

</__XML__OPT_Cmd_show_hardware_profile_buffer_monitor__readonly__>

</__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>

</monitor>

</buffer>

</profile>

</hardware>

</show>

</nf:data>

</nf:rpc-reply>

]]>]]>

```

### show hardware profile buffer monitor detail | xml

**XML :**

```

<show>
 <hardware>
 <profile>
 <buffer>
 <monitor>
 <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
 <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor__readonly__>
 <__readonly__>
 <cmd_name>Detail CLI</cmd_name>
 <cmd_issue_time>10/02/2001 10:58:58</cmd_issue_time>
 <TABLE_detail_entry>
 <ROW_detail_entry>
 <detail_util_name>Ethernet1/1</detail_util_name>
 <detail_util_state>Active</detail_util_state>
 </ROW_detail_entry>
 <ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:58</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 </ROW_detail_entry>
 </__readonly__>
 </__XML__OPT_Cmd_show_hardware_profile_buffer_monitor__readonly__>
 </__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
 </monitor>
 </buffer>
 </profile>
 </hardware>
</show>

```

```
<_XML_DIGIT3840k_util>0</_XML_DIGIT3840k_util>
<_XML_DIGIT4224k_util>0</_XML_DIGIT4224k_util>
<_XML_DIGIT4608k_util>0</_XML_DIGIT4608k_util>
<_XML_DIGIT4992k_util>0</_XML_DIGIT4992k_util>
<_XML_DIGIT5376k_util>0</_XML_DIGIT5376k_util>
<_XML_DIGIT5760k_util>0</_XML_DIGIT5760k_util>
<_XML_DIGIT6144k_util>0</_XML_DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:57</time_stamp>
 <_XML_DIGIT384k_util>0</_XML_DIGIT384k_util>
 <_XML_DIGIT768k_util>0</_XML_DIGIT768k_util>
 <_XML_DIGIT1152k_util>0</_XML_DIGIT1152k_util>
 <_XML_DIGIT1536k_util>0</_XML_DIGIT1536k_util>
 <_XML_DIGIT1920k_util>0</_XML_DIGIT1920k_util>
 <_XML_DIGIT2304k_util>0</_XML_DIGIT2304k_util>
 <_XML_DIGIT2688k_util>0</_XML_DIGIT2688k_util>
 <_XML_DIGIT3072k_util>0</_XML_DIGIT3072k_util>
 <_XML_DIGIT3456k_util>0</_XML_DIGIT3456k_util>
 <_XML_DIGIT3840k_util>0</_XML_DIGIT3840k_util>
 <_XML_DIGIT4224k_util>0</_XML_DIGIT4224k_util>
 <_XML_DIGIT4608k_util>0</_XML_DIGIT4608k_util>
 <_XML_DIGIT4992k_util>0</_XML_DIGIT4992k_util>
 <_XML_DIGIT5376k_util>0</_XML_DIGIT5376k_util>
 <_XML_DIGIT5760k_util>0</_XML_DIGIT5760k_util>
 <_XML_DIGIT6144k_util>0</_XML_DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:56</time_stamp>
 <_XML_DIGIT384k_util>0</_XML_DIGIT384k_util>
 <_XML_DIGIT768k_util>0</_XML_DIGIT768k_util>
 <_XML_DIGIT1152k_util>0</_XML_DIGIT1152k_util>
 <_XML_DIGIT1536k_util>0</_XML_DIGIT1536k_util>
 <_XML_DIGIT1920k_util>0</_XML_DIGIT1920k_util>
 <_XML_DIGIT2304k_util>0</_XML_DIGIT2304k_util>
 <_XML_DIGIT2688k_util>0</_XML_DIGIT2688k_util>
 <_XML_DIGIT3072k_util>0</_XML_DIGIT3072k_util>
 <_XML_DIGIT3456k_util>0</_XML_DIGIT3456k_util>
 <_XML_DIGIT3840k_util>0</_XML_DIGIT3840k_util>
 <_XML_DIGIT4224k_util>0</_XML_DIGIT4224k_util>
 <_XML_DIGIT4608k_util>0</_XML_DIGIT4608k_util>
 <_XML_DIGIT4992k_util>0</_XML_DIGIT4992k_util>
 <_XML_DIGIT5376k_util>0</_XML_DIGIT5376k_util>
 <_XML_DIGIT5760k_util>0</_XML_DIGIT5760k_util>
 <_XML_DIGIT6144k_util>0</_XML_DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:55</time_stamp>
 <_XML_DIGIT384k_util>0</_XML_DIGIT384k_util>
 <_XML_DIGIT768k_util>0</_XML_DIGIT768k_util>
 <_XML_DIGIT1152k_util>0</_XML_DIGIT1152k_util>
 <_XML_DIGIT1536k_util>0</_XML_DIGIT1536k_util>
 <_XML_DIGIT1920k_util>0</_XML_DIGIT1920k_util>
 <_XML_DIGIT2304k_util>0</_XML_DIGIT2304k_util>
 <_XML_DIGIT2688k_util>0</_XML_DIGIT2688k_util>
 <_XML_DIGIT3072k_util>0</_XML_DIGIT3072k_util>
 <_XML_DIGIT3456k_util>0</_XML_DIGIT3456k_util>
 <_XML_DIGIT3840k_util>0</_XML_DIGIT3840k_util>
 <_XML_DIGIT4224k_util>0</_XML_DIGIT4224k_util>
 <_XML_DIGIT4608k_util>0</_XML_DIGIT4608k_util>
 <_XML_DIGIT4992k_util>0</_XML_DIGIT4992k_util>
 <_XML_DIGIT5376k_util>0</_XML_DIGIT5376k_util>
 <_XML_DIGIT5760k_util>0</_XML_DIGIT5760k_util>
 <_XML_DIGIT6144k_util>0</_XML_DIGIT6144k_util>
```

```

</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:54</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>

```

### show hardware profile buffer monitor brief

XML :

```

show hardware profile buffer monitor brief | 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:mtc_usd_cli">
 <nf:data>
 <show>
 <hardware>
 <profile>
 <buffer>
 <monitor>
 <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
 <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor__readonly__>
 <__readonly__>
 <cmd_name>Brief CLI</cmd_name>
 <cmd_issue_time>03/21/2016 09:06:38</cmd_issue_time>
 <TABLE_ucst_hdr>
 <ROW_ucst_hdr>
 <ucst_hdr_util_name>Buffer Block 1</ucst_hdr_util_name>
 <ucst_hdr_1sec_util>0KB</ucst_hdr_1sec_util>
 <ucst_hdr_5sec_util>0KB</ucst_hdr_5sec_util>
 <ucst_hdr_60sec_util>N/A</ucst_hdr_60sec_util>
 <ucst_hdr_5min_util>N/A</ucst_hdr_5min_util>
 <ucst_hdr_1hr_util>N/A</ucst_hdr_1hr_util>
 <ucst_hdr_total_buffer>Total Shared Buffer Available = 5397 Kbytes
 </ucst_hdr_total_buffer>
 <ucst_hdr_class_threshold>Class Threshold Limit = 5130 Kbytes
 </ucst_hdr_class_threshold>
 </ROW_ucst_hdr>
 </TABLE_ucst_hdr>
 <TABLE_brief_entry>
 <ROW_brief_entry>
 <brief_util_name>Ethernet1/45</brief_util_name>
 <brief_1sec_util>0KB</brief_1sec_util>
 <brief_5sec_util>0KB</brief_5sec_util>
 <brief_60sec_util>N/A</brief_60sec_util>
 <brief_5min_util>N/A</brief_5min_util>
 <brief_1hr_util>N/A</brief_1hr_util>
 <brief_util_name>Ethernet1/46</brief_util_name>
 <brief_1sec_util>0KB</brief_1sec_util>

```





```
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
```

### show hardware profile latency monitor sampling

#### CLI

```
MTC-8(config)# show hardware profile latency monitor sampling

Sampling CLI issued at: 05/25/2016 04:19:54

Sampling interval: 20
```

#### XML

```
MTC-8(config)# show hardware profile latency monitor sampling | 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:mtc_usd_cli">

 <nf:data>

 <show>

 <hardware>

 <profile>

 <latency>

 <monitor>

 <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

 <__XML__OPT_Cmd_show_hardware_profile_latency_monitor__readonly__>

 <__readonly__>

 <cmd_issue_time>05/25/2016 04:20:06</cmd_issue_time>

 <device_instance>0</device_instance>

 <TABLE_sampling>

 <ROW_sampling>

 <sampling_interval>20</sampling_interval>

 </ROW_sampling>

 </TABLE_sampling>

 </__readonly__>

 </__XML__OPT_Cmd_show_hardware_profile_latency_monitor__readonly__>

 </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

 </monitor>

 </latency>
```

```

 </profile>
 </hardware>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

### show hardware profile latency monitor threshold

#### CLI

```
MTC-8(config)# show hardware profile latency monitor threshold
```

```
Sampling CLI issued at: 05/25/2016 04:20:53
```

```
Threshold Avg: 3000
```

```
Threshold Max: 300000
```

#### XML

```
MTC-8(config)# show hardware profile latency monitor threshold | 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:mtc_usd_cli">
 <nf:data>
 <show>
 <hardware>
 <profile>
 <latency>
 <monitor>
 <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
 <__XML__OPT_Cmd_show_hardware_profile_latency_monitor__readonly__>
 <__readonly__>
 <cmd_issue_time>05/25/2016 04:21:04</cmd_issue_time>
 <device_instance>0</device_instance>
 <TABLE_threshold>
 <ROW_threshold>
 <threshold_avg>3000</threshold_avg>
 <threshold_max>300000</threshold_max>
 </ROW_threshold>
 </TABLE_threshold>
 </__readonly__>
 </__XML__OPT_Cmd_show_hardware_profile_latency_monitor__readonly__>
 </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
 </monitor>
 </latency>
 </profile>
 </hardware>
 </show>
 </nf:data>
</nf:rpc-reply>

```

```
 </TABLE_threshold>
 </__readonly__>
 </__XML__OPT_Cmd_show_hardware_profile_latency_monitor__readonly__>
 </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
</monitor>
</latency>
</profile>
</hardware>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```



## CHAPTER 18

# Converting CLI Commands to Network Configuration Format

- [Information About XMLIN, on page 195](#)
- [Licensing Requirements for XMLIN, on page 195](#)
- [Installing and Using the XMLIN Tool, on page 196](#)
- [Converting Show Command Output to XML, on page 196](#)
- [Configuration Examples for XMLIN, on page 197](#)

## 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 20: 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.

## SUMMARY STEPS

1. switch# **xmlin**
2. switch(xmlin)# **configure terminal**
3. Configuration commands
4. (Optional) switch(config)(xmlin)# **end**
5. (Optional) switch(config-if-verify)(xmlin)# **show commands**
6. (Optional) switch(config-if-verify)(xmlin)# **exit**

## DETAILED STEPS

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	switch# <b>xmlin</b>	
<b>Step 2</b>	switch(xmlin)# <b>configure terminal</b>	Enters global configuration mode.
<b>Step 3</b>	Configuration commands	Converts configuration commands to NETCONF format.
<b>Step 4</b>	(Optional) switch(config)(xmlin)# <b>end</b>	Generates the corresponding <edit-config> request.  <b>Note</b> Enter the <b>end</b> command to finish the current XML configuration before you generate an XML instance for a <b>show</b> command.
<b>Step 5</b>	(Optional) switch(config-if-verify)(xmlin)# <b>show commands</b>	Converts <b>show</b> commands to NETCONF format.
<b>Step 6</b>	(Optional) switch(config-if-verify)(xmlin)# <b>exit</b>	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.

**SUMMARY STEPS**

1. switch# *show-command* | **xmlin**

**DETAILED STEPS**

**Procedure**

	Command or Action	Purpose
Step 1	switch# <i>show-command</i>   <b>xmlin</b>	Enters global configuration mode.  <b>Note</b> 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 ©) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php

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

# OpenConfig YANG

This section contains the following topics:

- [About OpenConfig YANG, on page 201](#)
- [Guidelines and Limitations for OpenConfig YANG, on page 201](#)
- [Understanding Deletion of BGP Routing Instance, on page 209](#)
- [Verifying YANG, on page 211](#)
- [Enabling OpenConfig Support, on page 211](#)

## 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. And, helping with moving from a pull model to a push model, with subscriptions and event update streaming.

Beginning with Cisco NX-OS Release 9.2(1), support is added across a broad range of functional areas. Those include BGP, OSPF, Interface L2 and L3, VRFs, VLANs, and TACACs.

CPU Utilization is the sum of user and kernel utilization values. This value is approximately equal to the sum of user and kernel percentages that are displayed in “show system resources” output.

The min, max, average, min time and max time values are calculated over a 30 second moving window. In this 30-second window, CPU utilization is calculated every 5 seconds. Min, max and average values are calculated over these six samples.

The time instant at which the minimum and maximum values occur within the last 30-second window, are displayed in epoch time format.

For additional information about OpenConfig YANG, see [About OpenConfig YANG](#).

For the OpenConfig models for Cisco NX-OS 9.2(1), see [YANG Models 9.2\(1\)](#). OpenConfig YANG models are grouped by Cisco NX-OS release, so when the Cisco NX-OS release number changes, the last digits in the URL change.

## Guidelines and Limitations for OpenConfig YANG

OpenConfig YANG has the following guidelines and limitations:

- For IPv4 and IPv6 addresses, you must provide the same operation for remove and delete for the IP address field (**oc-ip:ip** and **oc-ip:prefix\_length**).

For example:

```
oc-ip:ip: remove
oc-ip:prefix_length: remove
```

- Configuring BGP actions with **set med** and OSPF actions with metric in the same route-map via OpenConfig NETCONF is not recommended as the OSPF actions metric takes precedence over BGP **set med** property.

Use two different route-maps to set metrics under OSPF actions. Use **set-med** under BGP actions using separate route-maps.

We recommended that you do not change the metric of BGP actions to OSPF actions or OSPF actions to BGP actions of a route-map in a single payload.

- In order to have a valid BGP instance, an autonomous system (AS) number must be provided. Since there cannot be a default value for an AS number, any attempt to delete in NETCONF/OPENCONFIG `<asn>` without removing the BGP instance, results in the following highlighted error message:

```
764
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
 <nc:edit-config>
 <nc:target>
 <nc:running/>
 </nc:target>
 <nc:config>
 <network-instances xmlns="http://openconfig.net/yang/network-instance">
 <network-instance>
 <name>default</name>
 <protocols>
 <protocol>
 <identifier>BGP</identifier>
 <name>bgp</name>
 <bgp>
 <global>
 <config nc:operation="delete">
 <as>100</as>
 </config>
 </global>
 <neighbors>
 <neighbor>
 <neighbor-address>1.1.1.1</neighbor-address>
 <enable-bfd xmlns="http://openconfig.net/yang/bfd">
 <config>
 <enabled>true</enabled>
 </config>
 </enable-bfd>
 </neighbor>
 </neighbors>
 </bgp>
 </protocol>
 </protocols>
 </network-instance>
 </network-instances>
 </nc:config>
 </nc:edit-config>
</nc:rpc>

##
```

```

Received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
 <rpc-error>
 <error-type>protocol</error-type>
 <error-tag>operation-failed</error-tag>
 <error-severity>error</error-severity>
 <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst</error-message>
 <error-path>/config/network-instances</error-path>
 </rpc-error>
 <rpc-error>
 <error-type>protocol</error-type>
 <error-tag>operation-failed</error-tag>
 <error-severity>error</error-severity>
 <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst Commit Failed</error-message>
 <error-path>/config/network-instances</error-path>
 </rpc-error>
</rpc-reply>

```

- 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 will be appended after community-set-name if community-member is in the standard form (AS:NN).
- The `_exp` suffix will be appended after community-set-name if community-member is in the expanded form (regex):

```

<community-set>
 <community-set-name>oc_commset1d</community-set-name>
 <config>
 <community-set-name>oc_commset1d</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 a 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**) is not 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 9000 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>
 </subinterface>
 </subinterfaces>
 </interface>
 </interfaces>
 </config>
 </edit-config>
</rpc>
```



```

 <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.
- The following guidelines and limitations apply to OC-LACP:
  - Port-channel mode:

- OC-LACP enables configuring the port-channel mode on the port-channel interface. However, through the NXOS-CLI, the port-channel mode is configured on the member interface using channel-group mode active or passive.
- Although OC-LACP explicitly configures the port-channel mode on a port-channel interface, issuing the NX-OS **show running-config** command on a port-channel interface does not show the port-channel mode configuration for either empty or non-empty port-channels.
- Once a member is added to the port-channel, **show running interface ethernet <>** shows the port-channel mode configuration as a channel-group mode active or passive.




---

**Note** All port-channels that are created through OpenConfig should continue to be managed by OpenConfig.

---

- Port-channel interval rate:
  - Port channel interval can only be changed when members are in `shut` state.
  - The OC-LACP interval is per port-channel. The NX-OS LACP interval is per port-channel member. Because of this difference, the following behavior can be expected:
    - If you configure the port-channel interval through OpenConfig, all members in the port-channel get the same configuration applied to them.
    - If you configure the port-channel interval through OpenConfig and later a member is added to the port-channel, you must configure the interval again through OpenConfig for the configuration to be applied to the new member.
- System MAC ID:
  - In this release, Cisco NX-OS does not support `system-id-mac` per port-channel.
- Member-state data for the following is present only when a port is in `admin up` state:
  - LACP
  - Interface
  - Interfaces
  - Member
  - State
- OSPFv2 can send an error response when you attempt to add an interface through OpenConfig YANG. When the problem occurs, the interface is not added, and the RPC reply contains a "list merge failed" error as follows:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:39507023-8569-4cf8-869c-e19aaf76a260">
 <rpc-error>
 <error-type>protocol</error-type>
 <error-tag>operation-failed</error-tag>
 <error-severity>error</error-severity>
 </rpc-error>
</rpc-reply>
```

```

<error-message xml:lang="en">List Merge Failed: operation-failed</error-message>

<error-path>/network-instances/network-instance/protocols/protocol/cspf2/areas/area/interfaces/interface/ik</error-path>

 </rpc-error>
</rpc-reply>

```

- Queueing stats for Hig (ii) ports is not supported.
- You do not see the tx-packets, or bytes, and drop-packets per unicast, multicast, or broadcast queue. The stats that display in the OC response are a sum of the ucast, mcast, and bcast queues per qos-group.
- OpenConfig YANG does not support stats for a QoS policy that is applied at the VLAN level.
- The ingress queue drop count that can be retrieved through OC can be displayed at the slice/port/queue level depending on the platform.
- The following is the guideline and limitation for OpenConfig configurations for switchport, shut/no shut, MTU, and mac-address:
  - An ascii reload is required when configuring switchport, shut/no shut, MTU, and mac-address. Using a binary reload results in the configuration being lost.
- The following state containers are implemented for the OpenConfig ACL at interface-ref level:
  - /acl/interfaces/interface/interface-ref/state for acl/interfaces/state container.
  - acl/interfaces/interface/interface-ref/state/interface for read-onlyoc-if:interface leaf.
  - acl/interfaces/interface/interface-ref/state/subinterface for read-onlyoc-if:subinterface leaf.
- The following system config containers are implemented for domain-name, login-banner, and motd-banner models:
  - /system/config/domain-name for /top:System/top:dns-items/top:prof-items/top:Prof-list/top:dom-items/top:name container
  - system/config/login-banner for /top:System/top:userext-items/top:postloginbanner-items/top:message container
  - /system/config/motd-banner for /top:System/top:userext-items/top:preloginbanner-items/top:message container

## 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>
```

**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, and there is no equivalent single CLI configuration.

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
```

# Verifying YANG

Use the following commands to verify YANG settings: :

*Table 21: YANG Verification*

Command	Description
<code>show telemetry yang direct-path cisco-nxos-device</code>	Displays the paths which are supported.

## Enabling OpenConfig Support

To enable or disable OpenConfig support on the programmability agents (NETCONF, RESTCONF and gRPC), configure "[no] feature openconfig". For example:

```
switch(config)# feature netconf
switch(config)# feature restconf
switch(config)# feature grpc
switch(config)# feature openconfig
```



---

**Note** In previous releases, `mtx-openconfig-all` RPM was downloaded separately and installed. This method is deprecated in 10.2(2) release.

---





## CHAPTER 20

# XML Management Interface

---

This section contains the following topics:

- [About the XML Management Interface, on page 213](#)
- [Licensing Requirements for the XML Management Interface, on page 214](#)
- [Prerequisites to Using the XML Management Interface, on page 215](#)
- [Using the XML Management Interface, on page 215](#)
- [Information About Example XML Instances, on page 227](#)
- [Additional References, on page 233](#)

## About the XML Management Interface

### About the XML Management Interface

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

NETCONF is implemented with an XML Schema (XSD) that allows you to enclose device configuration elements within a remote procedure call (RPC) message. From within an RPC message, you select one of the NETCONF operations that matches the type of command that you want the device to execute. You can configure the entire set of CLI commands on the device with NETCONF. For information about using NETCONF, see the [Creating NETCONF XML Instances, on page 217](#) and [RFC 4741](#).

For more information about using NETCONF over SSH, see [RFC 4742](#).

This section includes the following topics:

- [NETCONF Layers, on page 213](#)
- [SSH xmlagent, on page 214](#)

### NETCONF Layers

The following are the NETCONF layers:

Table 22: NETCONF Layers

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

The following is a description of the four NETCONF layers:

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

## SSH xmlagent

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



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

NETCONF over SSH 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 the messages end, which keeps communication in sync.

The XML schemas that define XML configuration instances that you can use are described in the [Creating NETCONF XML Instances, on page 217](#) section.

## Licensing Requirements for the XML Management Interface

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



# Prerequisites to Using the XML Management Interface

The XML management interface has the following prerequisites:

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

## Using the XML Management Interface

This section describes how to manually configure and use the XML management interface. Use the XML management interface with the default settings on the device.

## Configuring SSH and the XML Server Options

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

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



---

**Note** The XML server timeout applies only to active sessions.

---

For more information about configuring SSH, see the Cisco NX-OS security configuration guide for your platform.

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

## Starting an SSH Session

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

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

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



---

**Note** The SSH command syntax can differ from the SSH software on the client PC.

---

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

- The SSH server is enabled on the device.
- The XML server max-sessions option is adequate to support the number of SSH connections to the device.

- The active XML server sessions on the device are not all in use.

## Sending the Hello Message

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

The following are sample hello messages from the server and the client.



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

### Hello Message from the server

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

### Hello Message from the Client

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

## Obtaining the XSD Files

### Procedure

- 
- Step 1** From your browser, navigate to the Cisco software download site at the following URL:  
<http://software.cisco.com/download/navigator.html>  
The Download Software page opens.
- Step 2** In the Select a Product list, choose **Switches > Data Center Switches > platform > model**.
- Step 3** If you are not already logged in as a registered Cisco user, you are prompted to log in now.
- Step 4** From the Select a Software Type list, choose **NX-OS XML Schema Definition**.

- Step 5** Find the desired release and click **Download**.
- Step 6** If you are requested, follow the instructions to apply for eligibility to download strong encryption software images. The Cisco End User License Agreement opens.
- Step 7** Click **Agree** and follow the instructions to download the file to your PC.

## Sending an XML Document to the XML Server

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

Follow these guidelines for this method:

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

## Creating NETCONF XML Instances

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

The following are the tags 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 `rpc`

All NETCONF XML instances must begin with the RPC request tag `<rpc>`. The example *RPC Request Tag* `<rpc>` shows the `<rpc>` element with its required **message-id** attribute. The message-id attribute is replicated in the `<rpc-reply>` and can be used to correlate requests and replies. The `<rpc>` node also contains the following XML namespace declarations:

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

### RPC Tag Request

```
<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

The following is an example of a configuration request.

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

\_\_XML\_\_MODE tags are used internally by the NETCONF agent. Some tags are present only as children of a certain \_\_XML\_\_MODE. By examining the schema file, you 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 23: NETCONF Operations in Cisco NX-OS**

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	<a href="#">NETCONF Close Session Instance, on page 227</a>
commit	Sets the running configuration to the current contents of the candidate configuration.	<a href="#">NETCONF Commit Instance - Candidate Configuration Capability, on page 232</a>
confirmed-commit	Provides parameters to commit the configuration for a specified time. If this operation is not followed by a commit operation within the confirm-timeout period, the configuration is reverted to the state before the confirmed-commit operation.	<a href="#">NETCONF Confirmed-commit Instance , on page 232</a>
copy-config	Copies the content of source configuration datastore to the target datastore.	<a href="#">NETCONF copy-config Instance, on page 228</a>
delete-config	Operation not supported.	—
edit-config	Configures features in the running configuration of the device. You use this operation for configuration commands.	<a href="#">NETCONF edit-config Instance, on page 228</a> <a href="#">NETCONF rollback-on-error Instance , on page 232</a>
get	Receives configuration information from the device. You use this operation for <b>show</b> commands. The source of the data is the running configuration.	<a href="#">Creating NETCONF XML Instances, on page 217</a>
get-config	Retrieves all or part of a configuration	<a href="#">NETCONF get-config Instance, on page 230</a>
kill-session	Closes the specified XML server session. You cannot close your own session. See the close-session NETCONF operation.	<a href="#">NETCONF Kill-session Instance, on page 228</a>

NETCONF Operation	Description	Example
lock	Allows the client to lock the configuration system of a device.	<a href="#">NETCONF Lock Instance, on page 230</a>
unlock	Releases the configuration lock that the session issued.	<a href="#">NETCONF unlock Instance, on page 231</a>
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	<a href="#">NETCONF validate Capability Instance , on page 233</a>

## Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See the [Obtaining the XSD Files, on page 216](#) section.

Using this schema, it is possible to build an XML instance. In the following examples, the relevant portions of the ncli.xsd schema file that was used to build [Creating NETCONF XML Instances, on page 217](#) is shown.

The following example shows XML device tags.

### 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>
```

The following example shows the server status device tags.

### server status Device Tags

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

```

<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>

```

The following example shows the device tag response.

### Device Tag Response

```

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

```



**Note** “\_\_XML\_\_OPT\_Cmd\_show\_xml\_\_readonly\_\_” is optional. This tag represents the response. For more information on responses, see the [RPC Response Tag](#), on page 226 section.

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

### XML Example

```

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

```

```

<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML_OPT_Cmd_show_xml__readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

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

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

## Extended NETCONF Operations

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

The following is an example of the tags 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

### Configuration CLI Commands Sent Through `<exec-command>`

```

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

```

The following is the response to the operation:

### Response to CLI Commands Sent Through `<exec-command>`

```

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

```

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



**show CLI Commands Sent Through <exec-command>**

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

The following is the response to the operation.

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

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

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

**Table 24: Tags**

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

Tag	Description
<cmd>	Contains the CLI command. A command can be a show or configuration command. Separate multiple configuration commands by using a semicolon “;”. Multiple show commands are not supported. You can send multiple configuration commands in different <cmd> tags as part of the same request. For more information, see the Example in <i>Configuration CLI Commands Sent Through &lt;exec-command&gt;</i> .

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

- <nf:ok>: All configure commands are executed successfully.
- <nf:rpc-error>: Some commands have failed. The operation stops on the first error, and the <nf:rpc-error> subtree provides more information on what configuration failed. Notice that any configuration that is executed before the failed command would have been applied to the running configuration.

The following example shows a failed configuration:

### Failed Configuration

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

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

The <rpc-reply> results from 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 a configuration and **show** command that are combined in the same instance.

## Combination of Configuration and show Commands

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

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

## Show CLI Commands Sent Through `<exec-command>`

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element><cmd></nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

## NETCONF Replies

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

This section contains the following topics:

- [RPC Response Tag, on page 226](#)
- [Interpreting Tags Encapsulated in the Data Tag, on page 226](#)

## RPC Response Tag

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

### RPC Response Elements

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

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

**Table 25: RPC Response Elements**

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

## Interpreting Tags Encapsulated in the Data Tag

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

### RPC-reply data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>
<show>
<interface>
<__XML__OPT_Cmd_show_interface_brief__readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
```

```

<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief__readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

<\_\_XML\_\_OPT.\*> and <\_\_XML\_\_BLK.\*> appear in responses and are sometimes used in requests. These tags are used by the NETCONF agent and are present in responses after the <\_\_readonly\_\_> tag. They are necessary in requests and should be added according to the schema file to reach the XML tag that represents the CLI command.

## Information About Example XML Instances

### Example XML Instances

This section provides the examples of the following XML instances:

- [NETCONF Close Session Instance, on page 227](#)
- [NETCONF Kill-session Instance, on page 228](#)
- [NETCONF copy-config Instance, on page 228](#)
- [NETCONF edit-config Instance, on page 228](#)
- [NETCONF get-config Instance, on page 230](#)
- [NETCONF Lock Instance, on page 230](#)
- [NETCONF unlock Instance, on page 231](#)
- [NETCONF Commit Instance - Candidate Configuration Capability, on page 232](#)
- [NETCONF Confirmed-commit Instance , on page 232](#)
- [NETCONF rollback-on-error Instance , on page 232](#)
- [NETCONF validate Capability Instance , on page 233](#)

### NETCONF Close Session Instance

The following example shows 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 example shows 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 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>]]>]]>
```

**NETCONF copy-config Instance**

The following example shows 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 example shows 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. 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.

## Edit-config: Delete Operation Request

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

```

<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>

```

### Response to edit-config: Delete Operation

```

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

```

## NETCONF get-config Instance

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

### Get-config Request to Retrieve the Entire Subtree

```

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

```

### Get-config Response with Results of the Query

```

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

```

## NETCONF Lock Instance

The following example shows the use of NETCONF lock operation.



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

### Lock Request

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

### Response to Successful Acquisition of Lock

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

### Response to Unsuccessful Attempt to Acquire the Lock

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

## NETCONF unlock Instance

The following example shows the use of the NETCONF unlock operation.

### unlock request

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

### response to unlock request

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

```
<ok/>
</rpc-reply>
```

## NETCONF Commit Instance - Candidate Configuration Capability

The following example shows the commit operation and the commit reply:

### 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 example shows the confirmed-commit operation and the 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 example shows the use of NETCONF rollback on error capability. The string `urn:ietf:params:netconf:capability:rollback-on-error:1.0` identifies the capability.

The following example shows 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 example shows the use of the NETCONF validate capability. The string `urn:ietf:params:netconf:capability:validate:1.0` identifies the 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.

### Standards

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

**RFCs**

<b>RFCs</b>	<b>Title</b>
<a href="#">RFC 4741</a>	NETCONF Configuration Protocol
<a href="#">RFC 4742</a>	Using the NETCONF Configuration Protocol over Secure Shell (SSH)



## PART I

# Model-Driven Programmability

- [Managing Components, on page 237](#)
- [Converting CLI Commands to Network Configuration Format, on page 243](#)
- [gNMI - gRPC Network Management Interface, on page 249](#)
- [gNOI-gRPC Network Operations Interface, on page 293](#)
- [Model Driven Telemetry, on page 301](#)





# CHAPTER 21

## Managing Components

---

- [About the Component RPM Packages, on page 237](#)
- [Preparing For Installation, on page 239](#)
- [Downloading Components from the Cisco Artifactory, on page 240](#)
- [Installing RPM Packages, on page 241](#)

## About the Component RPM Packages



**Note** Beginning with Cisco NX-OS Release 7.0(3)I6(2), the NX-OS Programmable Interface Base Component RPM packages (agents, the Cisco native model, most of the other required models, and infrastructure) are included in the Cisco NX-OS image. As a result, nearly all the required software is installed automatically when the image is loaded. This situation means that there is no need to download and install the bulk of the software from the Cisco Artifactory. The exception is the OpenConfig model, which is required. You must explicitly download the OpenConfig models from the Cisco Artifactory.

But, for Cisco NX-OS Release 7.0(3)I6(1) and earlier releases, if you need to upgrade, the following sections describing downloading and installing the packages are required.

---

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	-



Model Name	Model Rev	Model Ver	Package Name	Dependencies
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
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

RPM installation on the switch is performed in the 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 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
```

### Copying Files to the Device

You can use SCP to copy files to the device, using a command in this form.

**copy scp://username@source\_ip/path\_to\_agent\_rpm bootflash: vrf management**

Example:

```
Switch# copy scp://jdoe@192.0.20.123//myrpms/mtx-infra.1.0.0.r082616.x86_64.rpm bootflash: vrf management
```

### Displaying Installed NX-OS Programmable Interface RPMs

To show all installed NXOS Programmable Interface RPMs, issue the following command on the device:

```
bash-4.2# yum list installed | grep mtx
```

## Downloading Components from the Cisco Artifacts

The NX-OS Programmable Interface Component RPMs can be downloaded from the Cisco Artifacts 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 [Verify Device Readiness, on page 240](#), 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# dnf 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# dnf list installed | grep mtx
```

---



## CHAPTER 22

# Converting CLI Commands to Network Configuration Format

- [Information About XMLIN, on page 243](#)
- [Licensing Requirements for XMLIN, on page 243](#)
- [Installing and Using the XMLIN Tool, on page 244](#)
- [Converting Show Command Output to XML, on page 244](#)
- [Configuration Examples for XMLIN, on page 245](#)

## 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 26: 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.

## SUMMARY STEPS

1. switch# **xmlin**
2. switch(xmlin)# **configure terminal**
3. Configuration commands
4. (Optional) switch(config)(xmlin)# **end**
5. (Optional) switch(config-if-verify)(xmlin)# **show commands**
6. (Optional) switch(config-if-verify)(xmlin)# **exit**

## DETAILED STEPS

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	switch# <b>xmlin</b>	
<b>Step 2</b>	switch(xmlin)# <b>configure terminal</b>	Enters global configuration mode.
<b>Step 3</b>	Configuration commands	Converts configuration commands to NETCONF format.
<b>Step 4</b>	(Optional) switch(config)(xmlin)# <b>end</b>	Generates the corresponding <edit-config> request.  <b>Note</b> Enter the <b>end</b> command to finish the current XML configuration before you generate an XML instance for a <b>show</b> command.
<b>Step 5</b>	(Optional) switch(config-if-verify)(xmlin)# <b>show commands</b>	Converts <b>show</b> commands to NETCONF format.
<b>Step 6</b>	(Optional) switch(config-if-verify)(xmlin)# <b>exit</b>	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.

**SUMMARY STEPS**

1. switch# *show-command* | **xmlin**

**DETAILED STEPS****Procedure**

	Command or Action	Purpose
Step 1	switch# <i>show-command</i>   <b>xmlin</b>	Enters global configuration mode.  <b>Note</b> 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 ©) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php

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 23

# gNMI - gRPC Network Management Interface

This chapter contains the following topics:

- [About gNMI, on page 249](#)
- [gNMI RPC and SUBSCRIBE, on page 250](#)
- [Guidelines and Limitations for gNMI, on page 251](#)
- [Configuring gNMI, on page 254](#)
- [gNMI - gRPC Network Management Interface, on page 255](#)
- [Configuring Server Certificate, on page 256](#)
- [Generating Key/Certificate Examples, on page 257](#)
- [Generating Key/Certificate Examples , on page 258](#)
- [Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3\(3\) and Later, on page 258](#)
- [Enabling gNMI, on page 260](#)
- [Verifying gNMI, on page 261](#)
- [gRPC Client-Certificate-Authentication, on page 267](#)
- [Generating New Client Root CA Certificates, on page 267](#)
- [Configuring the Generated Root CA Certificates on NX-OS Device, on page 268](#)
- [Associating Trustpoints to gRPC, on page 268](#)
- [Validating the Certificate Details, on page 269](#)
- [Verifying the Connection using Client Certificate Authentication for any gNMI Clients, on page 270](#)
- [Clients, on page 270](#)
- [Sample DME Subscription - JSON Encoding, on page 271](#)
- [Sample DME Subscription - PROTO Encoding, on page 272](#)
- [Capabilities, on page 273](#)
- [Get, on page 277](#)
- [Set, on page 278](#)
- [Subscribe, on page 279](#)
- [Streaming Syslog, on page 282](#)
- [Troubleshooting, on page 288](#)

## About gNMI

gNMI uses gRPC (Google Remote Procedure Call) as its transport protocol.

Cisco NX-OS supports gNMI for dial-in subscription to telemetry applications running on switches. Although the past release supported telemetry events over gRPC, the switch pushed the telemetry data to the telemetry receivers. This method was called dial out.

With gNMI, applications can pull information from the switch. They subscribe to specific telemetry services by learning the supported telemetry capabilities and subscribing to only the telemetry services that it needs.

Cisco NX-OS Release 9.3(1) and later supports gNMI version 0.5.0.

Cisco NX-OS Release 9.3(5) and later supports gNMI version 0.5.0.

**Table 27: Supported gNMI RPCs**

gNMI RPC	Supported
Capabilities	Yes
Get	Yes
Set	Yes
Subscribe	Yes

## gNMI RPC and SUBSCRIBE

The NX-OS 9.3(1) release supports gNMI version 0.5.0. Cisco NX-OS Release 9.3(1) supports the following parts of gNMI version 0.5.0.

**Table 28: SUBSCRIBE Options**

Type	Sub Type	Supported?	Description
Once		Yes	Switch sends current values only once for all specified paths
Poll		Yes	Whenever the switch receives a Poll message, the switch sends the current values for all specified paths.
Stream	Sample	Yes	Once per stream sample interval, the switch sends the current values for all specified paths. The supported sample interval range is from 1 through 604800 seconds.  The default sample interval is 10 seconds.

Type	Sub Type	Supported?	Description
	On_Change	Yes	The switch sends current values as its initial state, but then updates the values only when changes, such as create, modify, or delete occur to any of the specified paths.
	Target_Defined	No	

### Optional SUBSCRIBE Flags

For the SUBSCRIBE option, some optional flags are available that modify the response to the options listed in the table. In release 9.3(1), the `updates_only` optional flag is supported, which is applicable to ON\_CHANGE subscriptions. If this flag is set, the switch suppresses the initial snapshot data (current state) that is normally sent with the first response.

The following flags are not supported:

- aliases
- allow\_aggregation
- extensions
- heart-beat interval
- prefix
- qos
- suppress\_redundant

## Guidelines and Limitations for gNMI

Following are the guidelines and limitations for gNMI:

- Beginning with Cisco NX-OS Release 9.3(5), Get and Set are supported.
- gNMI queries do not support wildcards in paths.
- gRPC traffic destined for a Nexus device will hit the control-plane policer (CoPP) in the default class. To limit the possibility of gRPC drops, configure a custom CoPP policy using the gRPC configured port in the management class.
- When you enable gRPC on both the management VRF and default VRF and later disable on the default VRF, the gNMI notifications on the management VRF stop working.

To enable gNMI notifications, gRPC must be enabled by entering the **feature gRPC** command. This enables gNMI notifications on the management VRF. If gNMI notifications are needed on the default VRF in addition to the management VRF, the **gRPC use-vrf default** command must be applied. When it is disabled on the default VRF by entering the **no gRPC use-vrf default** command, the gNMI notifications on the management VRF do not work.

As a workaround, disable gRPC completely by entering the **no feature grpc** command and reprovision it by entering the **feature grpc** command and any existing gRPC configuration commands. For example, **grpc certificate** or **grpc port**. You must also resubscribe to any existing notifications on the management VRF.

- When you attempt to subscribe an OpenConfig routing policy with a preexisting CLI configuration like the following, it returns empty values due to the current implementation of the OpenConfig model.

```
ip prefix-list bgp_v4_drop seq 5 deny 125.2.0.0/16 le 32
ipv6 prefix-list bgp_v6_drop seq 5 deny cafe:125:2::/48 le 128
```

using the xpath

```
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v4_drop]/config
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v6_drop]/config
```

- Only server certificate authentication takes place. The client certificate is not authenticated by the server.
- If the gRPC certificate is explicitly configured, after a reload with the saved startup configuration to a prior Cisco NX-OS 9.3(x) image, the gRPC feature does not accept connections. To confirm this issue, enter the **show grpc gnmi service statistics** command and the status line displays an error like the following:
 

```
Status: Not running - Initializing...Port not available or certificate invalid.
```

 Unconfigure and configure the proper certificate command to restore the service.
- Beginning with Cisco NX-OS Release 9.3(3), if you have configured a custom gRPC certificate, upon entering the **reload ascii** command the configuration is lost. It reverts to the default day-1 certificate. After entering the **reload ascii** command, the switch reloads. Once the switch is up again, you must reconfigure the gRPC custom certificate.




---

**Note** This applies when entering the **grpc certificate** command.

---

- The reachability in non-default VRF for gRPC is supported only over L3VNI's/EVPN and IP. However, reachability over MPLS in non-default VRF and VXLAN Flood and Learn is not supported.
- Use of origin, use\_models, or both, is optional for gNMI subscriptions.
- gNMI Subscription supports Cisco DME and Device YANG data models. Beginning with Cisco NX-OS Release 9.3(3), Subscribe supports the OpenConfig model.
- For Cisco NX-OS prior to 9.3(x), information about supported platforms, see *Platform Support for Programmability Features* in the guide for that release. Starting with Cisco NX-OS release 9.3(x), for information about supported platforms, see the [Nexus Switch Platform Matrix](#).
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12 MB maximum, the collected data is dropped. Applies to gNMI ON\_CHANGE mode only.

You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.

- Across all subscriptions, there is support of up to 150K aggregate MOs. Subscribing to more MOs can lead to collection data drops.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The feature supports Cisco DME and Device YANG data models. Openconfig YANG is not supported.
- The gRPC process that supports gNMI uses the HIGH\_PRIO control group, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The commands are not XMLized in this release.
  - The gRPC agent retains gNMI calls for a maximum of one hour after the call has ended.
  - If the total number of calls exceeds 2000, the gRPC agent purges ended calls based on the internal cleanup routine.
- Beginning with Cisco NX-OS Release 10.2(3)F, on change subscription of Device YANG ephemeral data (Accounting-log and Multicast) is supported.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of two gRPC servers on each switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load is not desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server process requests independent of the other. Requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

The following are the limitations for gNMI:

- multi-level wildcard "..." in path is not allowed
- wildcard '\*' in the top of the path is not allowed
- wildcard '\*' in key name is not allowed
- wildcard and value cannot be mixed in keys

The following table shows the wildcard support details for gNMI:

**Table 29: Wildcard Support for gNMI Requests**

Type of Request	Wildcard Support
gNMI GET	YES

Type of Request	Wildcard Support
gNMI SET	NO
gNMI SUBSCRIBE, ONCE	YES
gNMI SUBSCRIBE, POLL	YES
gNMI SUBSCRIBE, STREAM, SAMPLE	YES
gNMI SUBSCRIBE, STREAM, TARGET_DEFINED	YES
gNMI SUBSCRIBE, STREAM, ON_CHANGE	NO

## Configuring gNMI

Configure the gNMI feature through the **grpc gnmi** commands.

To import certificates used by the **grpc certificate** command onto the switch, see the [Installing Identity Certificates](#) section of the Cisco Nexus 3500 Series NX-OS Security Configuration Guide, Release 9.3(x).



**Note** When modifying the installed identity certificates or **grpc port** and **grpc certificate** values, the gRPC server might restart to apply the changes. When the gRPC server restarts, any active subscription is dropped and you must resubscribe.

### SUMMARY STEPS

1. **configure terminal**
2. **feature grpc**
3. (Optional) **grpc port** *port-id*
4. (Optional) **grpc certificate** *certificate-id*
5. **grpc gnmi max-concurrent-call** *number*

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enters global configuration mode.
<b>Step 2</b>	<b>feature grpc</b> <b>Example:</b>	Enables the gRPC agent, which supports the gNMI interface for dial-in.



	Command or Action	Purpose
	<pre>switch-1# feature grpc switch-1(config)#</pre>	
<b>Step 3</b>	(Optional) <b>grpc port</b> <i>port-id</i> <b>Example:</b> <pre>switch-1(config)# grpc port 50051</pre>	Configure the port number. The range of <i>port-id</i> is 1024–65535. 50051 is the default. <b>Note</b> This command is available beginning with Cisco NX-OS Release 9.3(3).
<b>Step 4</b>	(Optional) <b>grpc certificate</b> <i>certificate-id</i> <b>Example:</b> <pre>switch-1(config)# grpc certificate cert-1</pre>	Specify the certificate trustpoint ID. For more information, see the <a href="#">Installing Identity Certificates</a> section of the Cisco Nexus 30009000 Series NX-OS Security Configuration Guide, Release 9.3(x) for importing the certificate to the switch. <b>Note</b> This command is available beginning with Cisco NX-OS Release 9.3(3).
<b>Step 5</b>	<b>grpc gnmi max-concurrent-call</b> <i>number</i> <b>Example:</b> <pre>switch-1(config)# grpc gnmi max-concurrent-call 16 switch-1(config)#</pre>	Sets the limit of simultaneous dial-in calls to the gNMI server on the switch. Configure a limit from 1 through 16. The default limit is 8. The maximum value that you configure is for each VRF. If you set a limit of 16 and gNMI is configured for both management and default VRFs, each VRF supports 16 simultaneous gNMI calls. This command does not affect and ongoing or in-progress gNMI calls. Instead, gRPC enforces the limit on new calls, so any in-progress calls are unaffected and allowed to complete. <b>Note</b> The configured limit does not affect the gRPCConfigOper service.

## gNMI - gRPC Network Management Interface

This section defines a gRPC-based protocol for the modification and retrieval of the configuration from a target device, as well as, the control and generation of telemetry streams from a target device to a data collection system. The intention is that a single gRPC service definition can cover both configuration and telemetry - allowing a single implementation on the target, as well as a single NMS element to interact with the device via telemetry and configuration RPCs.

# Configuring Server Certificate

When you configured a TLS certificate and imported successfully onto the switch, the following is an example of the **show grpc gnmi service statistics** command output.

```
switch(config)# sh grpc gnmi service statistics

===== gRPC Endpoint
Vrf : management
Server address : [::]:50051

Cert notBefore : Nov 5 16:48:58 2015 GMT
Cert notAfter : Nov 5 16:48:58 2035 GMT
Client Root Cert notBefore : n/a
Client Root Cert notAfter : n/a

Max concurrent calls : 8
Listen calls : 1
Active calls : 0
KeepAlive Timeout : 120

Number of created calls : 1
Number of bad calls : 0

Subscription stream/once/poll : 0/0/0

Max gNMI::Get concurrent : 6
Max grpc message size : 25165824
gNMI Synchronous calls : 3
gNMI Synchronous errors : 3
gNMI Adapter errors : 3
gNMI Dtx errors : 0
```

gNMI communicates over gRPC and uses TLS to secure the channel between the switch and the client. The default hard-coded gRPC certificate is no longer shipped with the switch. The default behavior is a self-signed key and certificate which is generated on the switch as shown below with an expiration date of one day.

When the certificate is expired or failed to install successfully, you will see the 1-D default certificate. The following is an example of the **show grpc gnmi service statistics** command output.

```
#show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Wed Mar 11 19:43:01 PDT 2020
Cert notAfter : Thu Mar 12 19:43:01 PDT 2020

Max concurrent calls : 8
Listen calls : 1
Active calls : 0

Number of created calls : 1
Number of bad calls : 0

Subscription stream/once/poll : 0/0/0
```

With an expiration of one day, you can use this temporary certificate for quick testing. For long term a new key/certificate must be generated.

## Generating Key/Certificate Examples



**Note** Use this procedure for Cisco NX-OS Release 9.3(2) and earlier.

For Cisco NX-OS Release 9.3(3) and later, see

Typically, there are two possible scenarios:

1. The server and client can use a Self-Signed Certificate. Self-Signed Certificates are less secure and are not to be used in production environments.

**a.** Generating self-signed certificates on the switch:

1. Generating self-signed certificates on the switch:
  1. Login to the Bash shell:
  2. cd to the location where you want to store the key/cert
  3. switch# openssl req -x509 -newkey rsa:2048 -keyout self\_sign2048.key -out self\_sign2048.pem -days 365 -nodes
 

```

Generating a 2048 bit RSA private key
.....
.....
writing new private key to 'self_sign2048.key'

You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:US
string is too long, it needs to be less than 2 bytes long
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:San Jose
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Cisco
Systems
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:benton.cisco.com
Email Address []:
switch#
switch# ls -al | grep self
-rw-r--r-- 1 root root 912 Jun 6 15:16 self_sign2048.key
-rw-r--r-- 1 root root 952 Jun 6 15:16 self_sign2048.pem
switch#
```
2. Modify /etc/mtx.conf file. The mtx.conf file is not persistent after reloads but persistent through "no feature grpc" & "feature grpc"
3. Under the "grpc" section in the file add the key and cert location
 

```

[grpc]
key = /bootflash/self_sign2048.key
cert = /bootflash/self_sign2048.pem
```

4. Go to cli and run
  1. no feature grpc
  2. feature grpc
  3. show grpc internal gnmi service statistics
    1. Verify that the new cert and key are used by grpc process.
5. Optional: If you want the changes to be persistent across reloads
  1. Create a new file /etc/mtx.conf.user
  2. Add the following
 

```
[grpc]
key = /bootflash/self_sign2048.key
cert = /bootflash/self_sign2048.pem
```
  3. To get the mtx.conf file to accept the changes either
    1. Install, activate or de-activate a MTX RPM
    2. Or reload the box

2. You can generate a CA signed certificate.

## Generating Key/Certificate Examples

Follow these examples to generate Key/Certificates:

- [Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3\(3\) and Later, on page 258](#)

## Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3(3) and Later

The following is an example for generating key/certificate.



**Note** This task is an example of how a certificate can be generated on a switch. You can also generate a certificate in any Linux environment. In a production environment, you should consider using a CA signed certificate.

For more information on generating identity certificates, see the [Installing Identity Certificates](#) section of the *Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)*.

### Procedure

**Step 1** Generate the selfsigned key and pem files.

```
switch# run bash sudo su
bash-4.3# openssl req -x509 -newkey rsa:2048 -keyout self_sign2048.key -out self_sign2048.pem -days
365 -nodes
```

a) switch# openssl req -x509 -newkey rsa:2048 -keyout self\_sign2048.key -out self\_sign2048.pem -days 365 -nodes

**Step 2** After generating the key and pem files, you must bundle the key and pem files for use in the trustpoint CA Association.

```
switch# run bash sudo su
bash-4.3# cd /bootflash/
bash-4.3# openssl pkcs12 -export -out self_sign2048.pfx -inkey self_sign2048.key -in self_sign2048.pem
-certfile self_sign2048.pem -password pass:Ciscolab123!
bash-4.3# exit
```

**Step 3** Set up the trustpoint CA Association by inputting in the pkcs12 bundle into the trustpoint.

```
switch(config)# crypto ca trustpoint mytrustpoint
switch(config-trustpoint)# crypto ca import mytrustpoint pkcs12 self_sign2048.pfx Ciscolab123!
```

**Step 4** Verify the setup.

```
switch(config)# show crypto ca certificates
Trustpoint: mytrustpoint
certificate:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov 5 16:48:58 2015 GMT
notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient

CA certificate 0:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov 5 16:48:58 2015 GMT
notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
```

**Step 5** Configure gRPC to use the trustpoint.

```
switch(config)# grpc certificate mytrustpoint
switch(config)# show run grpc

!Command: show running-config grpc
!Running configuration last done at: Thu Jul 2 12:24:02 2020
!Time: Thu Jul 2 12:24:05 2020

version 9.3(5) Bios:version 05.38
feature grpc

grpc gnmi max-concurrent-calls 16
grpc use-vrf default
grpc certificate mytrustpoint
```

**Step 6** Verify gRPC is now using the certificate.

```
switch# show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf : management
```

```

Server address : [::]:50051

Cert notBefore : Nov 5 16:48:58 2015 GMT
Cert notAfter : Nov 5 16:48:58 2035 GMT

Max concurrent calls : 16
Listen calls : 1
Active calls : 0

Number of created calls : 953
Number of bad calls : 0

Subscription stream/once/poll : 476/238/238

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 10
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0

```

## Enabling gNMI

This procedure starts the gNMI agent which runs on the management VRF and port 50051.

You can configure it to run on the additional default VRF or configure the maximal concurrent connection number.

### Before you begin

You might need to configure the key/cert. If you do not, the server generates a pair of temporary key/certs under `/opt/mtx/etc/grpc.pem` and `/opt/mtx/etc/grpc.key`. You need to download the key/cert to its clients. It is valid for one-day use.

### SUMMARY STEPS

1. **configure terminal**
2. **feature grpc**
3. **grpc gnmi max-concurrent-calls *number***
4. **grpc use-vrf default**

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
Step 1	<b>configure terminal</b>  <b>Example:</b> switch# <b>configure terminal</b>	Enters global configuration mode.

	Command or Action	Purpose
Step 2	<b>feature grpc</b> <b>Example:</b> <pre>switch(config)# feature grpc</pre>	Enables the gRPC agent, which supports the gNMI interface.
Step 3	<b>grpc gnmi max-concurrent-calls <i>number</i></b> <b>Example:</b> <pre>switch(config)# grpc gnmi max-concurrent-calls 16</pre>	<p>Sets the limit of simultaneous dial-in calls to the gNMI server on the switch. Configure a limit from 1 through 16. The default limit is 8.</p> <p>The maximum value that you configure is for each VRF. If you set a limit of 16 and gNMI is configured for both the management and default VRF, each VRF supports 16 simultaneous gNMI calls.</p> <p>This command does not affect ongoing or in-progress gNMI calls. Instead, gRPC enforces the limit on new calls. Any in-progress calls are unaffected and allowed to complete.</p> <p><b>Note</b> The configured limit does not affect the gRPCConfigOper service.</p>
Step 4	<b>grpc use-vrf default</b> <b>Example:</b> <pre>switch(config)# grpc use-vrf default</pre>	<p>Enables the gRPC agent to accept incoming (dial-in) RPC requests from the default VRF. This step enables the default VRF to process incoming RPC requests. By default, the management VRF processes incoming RPC requests when the gRPC feature is enabled.</p> <p><b>Note</b> Both VRFs process requests individually, so that requests do not cross between VRFs.</p>

## Verifying gNMI

To verify the gNMI configuration, enter the following command:

Command	Description
<b>show grpc gnmi service statistics</b>	<p>Displays a summary of the agent running status, respectively for the management VRF, or the default VRF (if configured). It also displays:</p> <ul style="list-style-type: none"> <li>• Basic overall counters</li> <li>• Certificate expiration time</li> </ul> <p><b>Note</b> If the certificate is expired, the agent cannot accept requests.</p>

Command	Description
<b>show grpc gnmi rpc summary</b>	Displays the following: <ul style="list-style-type: none"><li>• Number of capability RPCs received.</li><li>• Capability RPC errors.</li><li>• Number of Get RPCs received.</li><li>• Get RPC errors.</li><li>• Number of Set RPCs received.</li><li>• Set RPC errors.</li><li>• More error types and counts.</li></ul>



Command	Description
<p><b>show grpc gnmi transactions</b></p>	<p>The <b>show grpc gnmi transactions</b> command is the most dense and contains considerable information. It is a history buffer of the most recent 50 gNMI transactions that are received by the switch. As new RPCs come in, the oldest history entry is removed from the end. The following explains what is displayed:</p> <ul style="list-style-type: none"> <li>• RPC – This shows the type of RPC that was received (Get, Set, Capabilities)</li> <li>• DataType – For a Get only. Has values ALL, CONFIG, and STATE.</li> <li>• Session – shows the unique session-id that is assigned to this transaction. It can be used to correlate data that is found in other log files.</li> <li>• Time In -- shows timestamp of when the RPC was received by the gNMI handler.</li> <li>• Duration – time delta in ms from receiving the request to giving response.</li> <li>• Status – the status code of the operation returned to the client (0 = Success, !0 == error).</li> </ul> <p>This section is data that is kept per path within a single gNMI transaction. For example, a single Get or Set</p> <ul style="list-style-type: none"> <li>• subtype – for a Set RPC, shows the specific operation that is requested per path (Delete, Update, Replace). For Get, there is no subtype.</li> <li>• dtx – shows that this path was processed in DTX “fast” path or not. A dash ‘-’ means no, an asterisk ‘*’ means yes.</li> <li>• st – Status for this path. The meaning is as follows: <ul style="list-style-type: none"> <li>• OK: path is valid and processed by infra successfully.</li> <li>• ERR: path is either invalid or generated error by infra</li> <li>• --: path not processed yet, might or might not be valid and has not been sent to infra yet.</li> </ul> </li> <li>• path – the path</li> </ul>

**show grpc gnmi service statistics Example**

```

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter : Nov 20 19:05:24 2033 GMT

Max concurrent calls : 8
Listen calls : 1
Active calls : 0

Number of created calls : 1
Number of bad calls : 0

Subscription stream/once/poll : 0/0/0

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 74
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0

```

**show grpc gnmi rpc summary Example**

```

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Mar 31 20:55:02 2020 GMT
Cert notAfter : Apr 1 20:55:02 2020 GMT

Capability rpcs : 1
Capability errors : 0
Get rpcs : 53
Get errors : 19
Set rpcs : 23
Set errors : 8
Resource Exhausted : 0
Option Unsupported : 6
Invalid Argument : 18
Operation Aborted : 1
Internal Error : 2
Unknown Error : 0

RPC Type State Last Activity Cnt Req Cnt Resp Client

Subscribe Listen 04/01 07:39:21 0 0

```

**show grpc gnmi transactions Example**

```

=====
gRPC Endpoint

```

=====

Vrf : management  
Server address : [::]:50051

Cert notBefore : Mar 31 20:55:02 2020 GMT  
Cert notAfter : Apr 1 20:55:02 2020 GMT

RPC	DataType	Session	Time In	Duration(ms)	Status
Set	-	2361443608	04/01 07:43:49	173	0
subtype: dtx: st: path:					
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo789]		
Set	-	2293989720	04/01 07:43:45	183	0
subtype: dtx: st: path:					
Replace	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo6]		
Set	-	2297110560	04/01 07:43:41	184	0
subtype: dtx: st: path:					
Update	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo7]		
Set	-	0	04/01 07:43:39	0	10
Set	-	3445444384	04/01 07:43:33	3259	0
subtype: dtx: st: path:					
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo789]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo790]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo791]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo792]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo793]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo794]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo795]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo796]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo797]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo798]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo799]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo800]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo801]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo802]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo803]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo804]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo805]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo806]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo807]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo808]		
Set	-	2297474560	04/01 07:43:26	186	0
subtype: dtx: st: path:					
Update	-	OK	/System/ipv4-items/inst-items/dom-items/Dom-list[name=foo]/rt-items/Route-list[prefix=0.0.0.0/0]/nh-items/NextHop-list[nhAddr=192.168.1.1/32][nhVrf=foo][nhIf=unspecified]/tag		
Set	-	2294408864	04/01 07:43:17	176	13
subtype: dtx: st: path:					
Delete	-	ERR	/System/intf-items/lb-items/LbRtdIf-list/descr		
Set	-	0	04/01 07:43:11	0	3
subtype: dtx: st: path:					
Update	-	--	/System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr		
Update	-	ERR	/system/processes		

```

Set - 2464255200 04/01 07:43:05 708 0
subtype: dtx: st: path:
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo2]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo777]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo778]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo779]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo780]
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr

Set - 3491213208 04/01 07:42:58 14 0
subtype: dtx: st: path:
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr

Set - 3551604840 04/01 07:42:54 35 0
subtype: dtx: st: path:
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo1]

Set - 2362201592 04/01 07:42:52 13 13
subtype: dtx: st: path:
Delete - ERR /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/lbrtdif-items
/operSt

Set - 0 04/01 07:42:47 0 3
subtype: dtx: st: path:
Delete - ERR /System/*

Set - 2464158360 04/01 07:42:46 172 3
subtype: dtx: st: path:
Delete - ERR /system/processes/shabang

Set - 2295440864 04/01 07:42:46 139 3
subtype: dtx: st: path:
Delete - ERR /System/invalid/path

Set - 3495739048 04/01 07:42:44 10 0

Get ALL 3444580832 04/01 07:42:40 3 0
subtype: dtx: st: path:
- - OK /System/bgp-items/inst-items/disPolBatch

Get ALL 0 04/01 07:42:36 0 3
subtype: dtx: st: path:
- - -- /system/processes/process[pid=1]

Get ALL 3495870472 04/01 07:42:36 2 0
subtype: dtx: st: path:
- * OK /system/processes/process[pid=1]

Get ALL 2304485008 04/01 07:42:36 33 0
subtype: dtx: st: path:
- * OK /system/processes

Get ALL 2464159088 04/01 07:42:36 251 0
subtype: dtx: st: path:
- - OK /system

```

```

Get ALL 2293232352 04/01 07:42:35 258 0
subtype: dtx: st: path:
- - OK /system

Get ALL 0 04/01 07:42:33 0 12
subtype: dtx: st: path:
- - -- /intf-items

```

## gRPC Client-Certificate-Authentication

Beginning with 10.1(1) release, an additional authentication method is provided for gRPC. gRPC services prior to 10.1(1) release supported only the server certificate. Starting from 10.1(1), authentication is enhanced to add support for client certificate as well so that gRPC allows to verify both server certificate and client certificate. This enhancement provides password-less authentication for different Clients.

## Generating New Client Root CA Certificates

The following is the example for generating a new certificate to the client root:

- Trusted Certificate Authorities (CA)

Perform the following steps when you use a trusted CA such as a DigiCert:

### SUMMARY STEPS

1. Download the CA certificate file.
2. Import to NX-OS using the steps in [Cisco NX-OS Security Configuration Guide](#).

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
<b>Step 1</b>	Download the CA certificate file.	
<b>Step 2</b>	Import to NX-OS using the steps in <a href="#">Cisco NX-OS Security Configuration Guide</a> .	<ul style="list-style-type: none"> <li>• To create a trustpoint label, use steps in <a href="#">Creating a Trustpoint CA Association</a></li> <li>• To authenticate the trustpoint using the trusted CA certificates, use steps in <a href="#">Authenticating the CA</a>.</li> </ul> <p><b>Note</b> Use the CA Certificate from cat [CA_cert_file].</p>

# Configuring the Generated Root CA Certificates on NX-OS Device

When you have generated a new certificate to the client root successfully, following are the sample commands to configure them in the switch, and their output.

```
switch(config)# crypto ca trustpoint my_client_trustpoint
enticate my_client_trustpoint
switch(config-trustpoint)# crypto ca authenticate my_client_trustpoint
input (cut & paste) CA certificate (chain) in PEM format;
end the input with a line containing only END OF INPUT :
-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJAJLisBKCGjQOMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
BAYTA1VTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIU2FuIEpvc2UxDjAMBgNVBAoM
BUNpc2NvMB4XDTEwMTAxNDIwNTYyN1oXDTQwMTAwOTIwNTYyN1owPTElMAkGA1UE
BhMCVVMxCzAJBgNVBAGMAkNBMRERDwYDVQQHDAhTYW4gSm9zZTEOMAwGA1UECgWF
Q2l2Y28wggeiEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDEX7qZ2EdogZU4
EW0NSpB3EjY0nSlFLOw/iLKsxfIiQJD0Qhaw16fDnnYZj6vzWEa0ls8canqHCXQl
gUyxFOdGDxa6neQFTqLowSA6UCSQA+eenN2PIpMOjfdFpaPiHu3mmcTI1xP39Ti3
/y548NNORSepApBNkZ1rJSB6Cu9AIFMZgrZXFqDKBGSUOf/CPnvIDZeLcun+zpUu
CxJLA76Et4buPMysuRqMGHIX8CYw8MtjmuCuCTHXNN31ghhgpfXfrW/69pykjU3R
YOrwlsUkvYQhtefHuTHBmqym7MFoBEchwrlC5YTduDzmOvtkhsmpogRe3BiIBx45
AnZdtdilAgMBAAGjUzBRMB0GAlUdDgQWBBSh3IqRrm+mtB5GNsoLXFb3bAVg5Taf
BgNVHSMEGDAWgBSh3IqRrm+mtB5GNsoLXFb3bAVg5TAPBgNVHRMBAf8EBTADAQH/
MA0GCSqGSIb3DQEBCwUAA4IBAQA4Fpc6lRkzBGJQ/7oK1FNcTX/YXkneXdk7Zrj
8W0RS0Khxgke97d2Cw15P5reXO27kvXsnsz/VZn7JYGUvGS1xTlcCb6x6wNBr4Qr
t9qDBu+LykwqNOFe4VCAv6e4cMXNbh2wHBVS/NSoWnM2FGZ10VppjEGFm6OM+N6z
8n4/rWslfWFbn7T7xHH+N10Ffc+8q8h37opyCnb0ILj+a4rnyus8xXJPQb05DfJe
ahPNfdEsXKDOWNrSDtmKwtWDqdtjSQc4xioKHoshnNgWBjbovP1MQ64UrajBycwv
z9snWBm6p9SdTsV92YwF1tRGUqpcI9olsBgH7FUVU1hmHDWE
-----END CERTIFICATE-----
END OF INPUT
Fingerprint(s): SHA1 Fingerprint=0A:61:F8:40:A0:1A:C7:AF:F2:F7:D9:C7:12:AE:29:15:52:9D:D2:AE
```

```
Do you accept this certificate? [yes/no]:yes
switch(config)#
```

NOTE: Use the CA Certificate from the .pem file content.

```
switch# show crypto ca certificates
Trustpoint: my_client_trustpoint
CA certificate 0:
subject=C = US, ST = CA, L = San Jose, O = Cisco
issuer=C = US, ST = CA, L = San Jose, O = Cisco
serial=B7E30B8F4168FB87
notBefore=Oct 1 17:29:47 2020 GMT
notAfter=Sep 26 17:29:47 2040 GMT
SHA1 Fingerprint=E4:91:4E:D4:41:D2:7D:C0:5A:E8:F7:2D:32:81:B3:37:94:68:89:10
purposes: sslserver sslclient
```

## Associating Trustpoints to gRPC

When you have configured a new certificate to the client root successfully, the following is the output example for associating trustpoints to gRPCs on the switch:



**Note** Configuring or removing the root certificate for client authentication will cause gRPC process to restart.

```
switch(config)# feature grpc

switch(config)# grpc client root certificate my_client_trustpoint
switch(config)# show run grpc

!Command: show running-config grpc
!Running configuration last done at: Wed Dec 16 20:18:35 2020
!Time: Wed Dec 16 20:18:40 2020

version 10.1(1) Bios:version N/A
feature grpc

grpc gnmi max-concurrent-calls 14
grpc use-vrf default
grpc certificate my_trustpoint
grpc client root certificate my_client_trustpoint
grpc port 50003
```

## Validating the Certificate Details

When you have successfully associated the trustpoints to gRPC on the switch, the following is the output example for validating the certificate details:

```
switch# show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50003

Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter : Nov 20 19:05:24 2033 GMT
Client Root Cert notBefore : Oct 1 17:29:47 2020 GMT
Client Root Cert notAfter : Sep 26 17:29:47 2040 GMT

Max concurrent calls : 14
Listen calls : 1
Active calls : 0

Number of created calls : 1
Number of bad calls : 0

Subscription stream/once/poll : 0/0/0

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 0
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

# Verifying the Connection using Client Certificate Authentication for any gNMI Clients

The client certificate requests with a private key (pkey) and ca chain (cchain). The password is now optional.

Performing GetRequest, encoding = JSON to 172.19.199.xxx with the following gNMI Path

```

[elem {
 name: "System"
}
elem {
 name: "bgp-items"
}
]
```

The GetResponse is below

```

notification {
 timestamp: 1608071208072199559
 update {
 path {
 elem {
 name: "System"
 }
 elem {
 name: "bgp-items"
 }
 }
 val {
 json_val: ""
 }
 }
}
```

For removing trustpoint reference from gRPC (no command) use the following command:

```
[no] grpc client root certificate <my_client_trustpoints>
switch(config)# no grpc client root certificate my_client_trustpoint
```

The command will remove the trustpoint reference only from gRPC agent, but the trustpoints CA certificates will NOT be removed. Connections that use client certificate authentication to gRPC server on switch will not establish, but basic authentication with username and password will go through.




---

**Note** If the client's certificate is signed by intermediate CAs, but not directly by the root CA that is imported from the above config, the grpc client needs to supply the full cert chain, including the user, intermediate CA cert, and the root CA cert.

---

## Clients

There are available clients for gNMI. One such client is located at [https://github.com/influxdata/telegraf/tree/master/plugins/inputs/cisco\\_telemetry\\_gnmi](https://github.com/influxdata/telegraf/tree/master/plugins/inputs/cisco_telemetry_gnmi).



## Sample DME Subscription - JSON Encoding

```

gnmi-console --host <iip> --port 50051 -u <user> -p <pass> --tls --
operation=Subscribe --rpc bl_payload/once/01_subscribe_bgp_dme.json

[Subscribe]-----
Reading from file ' bl_payload/once/01_subscribe_bgp_dme.json '

Generating request : 1 -----
Comment : ONCE request
Delay : 2 sec(s) ...
Delay : 2 sec(s) DONE
subscribe {
subscription {
path {
origin: "DME"
elem {
name: "sys"
}
elem {
name: "bgp"
}
}
}
mode: ONCE
use_models {
name: "DME"
organization: "Cisco Systems, Inc."
version: "1.0.0"
}
}

Received response 1 -----
update {
timestamp: 1549061991079
update {
path {
elem {
name: "sys"
}
elem {
name: "bgp"
}
}
}
"\\" } }] } }] } }] } }] } } }"
}
duplicates: 1093487956
}
}
/Received -----
Received response 2 -----
sync_response: true
/Received -----
(_gnmi) [root@tm-ucs-1 gnmi-console]#

```

•

## Sample DME Subscription - PROTO Encoding

```

gnmi-console --host >iip> --port 50051 -u <user> -p <pass> --tls --
operation=Subscribe --rpc /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json

[Subscribe]-----
Reading from file ' /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json '
Wed Jun 26 11:49:17 2019
Generating request : 1 -----
Comment : ONCE request
Delay : 2 sec(s) ...
Delay : 2 sec(s) DONE
subscribe {
 subscription {
 path {
 origin: "DME"
 elem {
 name: "sys"
 }
 elem {
 name: "bgp"
 }
 }
 mode: SAMPLE
 }
 mode: ONCE
 use_models {
 name: "DME"
 organization: "Cisco Systems, Inc."
 version: "1.0.0"
 }
 encoding: PROTO
}
Wed Jun 26 11:49:19 2019
Received response 1 -----
update {
 timestamp: 1561574967761
 prefix {
 elem {
 name: "sys"
 }
 elem {
 name: "bgp"
 }
 }
 update {
 path {
 elem {
 }
 elem {
 name: "version_str"
 }
 }
 val {
 string_val: "1.0.0"
 }
 }
 update {
 path {
 elem {
 }
 }
 elem {
 }
 }
}

```

```

name: "node_id_str"
}
}
val {
string_val: "n9k-tm2"
}
}
update {
path {
elem {
}
}
elem {
name: "encoding_path"
}
}
}
val {
string_val: "sys/bgp"
}
}
update {
path {
elem {
}
}
elem {
/Received -----
Wed Jun 26 11:49:19 2019
Received response 2 -----
sync_response: true
/Received -----
(_gnmi) [root@tm-ucs-1 gnmi-console]#

```

# Capabilities

## About Capabilities

The Capabilities RPC returns the list of capabilities of the gNMI service. The response message to the RPC request includes the gNMI service version, the versioned data models, and data encodings supported by the server.

## Guidelines and Limitations for Capabilities

Following are the guidelines and limitations for Capabilities:

- Beginning with Cisco NX-OS Release 9.3(3), Capabilities supports the OpenConfig model.
- The gNMI feature supports Subscribe and Capability as options of the gNMI service.
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12-MB maximum, the collected data is dropped.

You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.

- All paths within the same subscription request must have the same sample interval. If the same path requires different sample intervals, create multiple subscriptions.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The feature supports Cisco DME and Device YANG data models. Openconfig YANG is not supported.
- The gRPC process that supports gNMI uses the HIGH\_PRIO cgroup, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The commands are not XMLized in this release.
  - The gRPC agent retains gNMI calls for a maximum of 1 hour after the call has ended.
  - If the total number of calls exceeds 2000, the gRPC agent purges ended calls based an internal cleanup routine.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of 2 gRPC servers on each switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load might not be desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server processes requests independent of the other, and requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

## Example Client Output for Capabilities

In this example, all the OpenConfig model RPMs have been installed on the switch.

The following is an example of client output for Capabilities.

```
hostname user$./gnmi_cli -a 172.19.193.166:50051 -ca_crt ./grpc.pem -insecure -capabilities
supported_models: <
 name: "Cisco-NX-OS-device"
 organization: "Cisco Systems, Inc."
 version: "2019-11-13"
>
supported_models: <
 name: "openconfig-acl"
 organization: "OpenConfig working group"
 version: "1.0.0"
>
supported_models: <
 name: "openconfig-bgp-policy"
 organization: "OpenConfig working group"
 version: "4.0.1"
```

```
>
supported_models: <
 name: "openconfig-interfaces"
 organization: "OpenConfig working group"
 version: "2.0.0"
>
supported_models: <
 name: "openconfig-if-aggregate"
 organization: "OpenConfig working group"
 version: "2.0.0"
>
supported_models: <
 name: "openconfig-if-ethernet"
 organization: "OpenConfig working group"
 version: "2.0.0"
>
supported_models: <
 name: "openconfig-if-ip"
 organization: "OpenConfig working group"
 version: "2.3.0"
>
supported_models: <
 name: "openconfig-if-ip-ext"
 organization: "OpenConfig working group"
 version: "2.3.0"
>
supported_models: <
 name: "openconfig-lacp"
 organization: "OpenConfig working group"
 version: "1.0.2"
>
supported_models: <
 name: "openconfig-lldp"
 organization: "OpenConfig working group"
 version: "0.2.1"
>
supported_models: <
 name: "openconfig-network-instance"
 organization: "OpenConfig working group"
 version: "0.11.1"
>
supported_models: <
 name: "openconfig-network-instance-policy"
 organization: "OpenConfig working group"
 version: "0.1.1"
>
supported_models: <
 name: "openconfig-ospf-policy"
 organization: "OpenConfig working group"
 version: "0.1.1"
>
supported_models: <
 name: "openconfig-platform"
 organization: "OpenConfig working group"
 version: "0.12.2"
>
supported_models: <
 name: "openconfig-platform-cpu"
 organization: "OpenConfig working group"
 version: "0.1.1"
>
supported_models: <
 name: "openconfig-platform-fan"
 organization: "OpenConfig working group"
```

```

 version: "0.1.1"
 >
 supported_models: <
 name: "openconfig-platform-linecard"
 organization: "OpenConfig working group"
 version: "0.1.1"
 >
 supported_models: <
 name: "openconfig-platform-port"
 organization: "OpenConfig working group"
 version: "0.3.2"
 >
 supported_models: <
 name: "openconfig-platform-psu"
 organization: "OpenConfig working group"
 version: "0.2.1"
 >
 supported_models: <
 name: "openconfig-platform-transceiver"
 organization: "OpenConfig working group"
 version: "0.7.0"
 >
 supported_models: <
 name: "openconfig-relay-agent"
 organization: "OpenConfig working group"
 version: "0.1.0"
 >
 supported_models: <
 name: "openconfig-routing-policy"
 organization: "OpenConfig working group"
 version: "2.0.1"
 >
 supported_models: <
 name: "openconfig-spanning-tree"
 organization: "OpenConfig working group"
 version: "0.2.0"
 >
 supported_models: <
 name: "openconfig-system"
 organization: "OpenConfig working group"
 version: "0.3.0"
 >
 supported_models: <
 name: "openconfig-telemetry"
 organization: "OpenConfig working group"
 version: "0.5.1"
 >
 supported_models: <
 name: "openconfig-vlan"
 organization: "OpenConfig working group"
 version: "3.0.2"
 >
 supported_models: <
 name: "DME"
 organization: "Cisco Systems, Inc."
 >
 supported_models: <
 name: "Cisco-NX-OS-Syslog-oper"
 organization: "Cisco Systems, Inc."
 version: "2019-08-15"
 >
 supported_encodings: JSON
 supported_encodings: PROTO
 gNMI_version: "0.5.0"

```

```
hostname user$
```

# Get

## About Get

The purpose of the Get RPC is to allow a client to retrieve a snapshot of the data tree from the device. Multiple paths may be requested in a single request. A simplified form of XPATH according to the gNMI Path Conventions, [Schema path encoding conventions for gNMI](#) are used for the path.

For detailed information on the Get operation, refer to the Retrieving Snapshots of State Information section in the gNMI specification: [gRPC Network Management Interface \(gNMI\)](#)

## Guidelines and Limitations for Get

The following are guidelines and limitations for Get and Set:

- `GetRequest.encoding` supports only JSON.
- For `GetRequest.type`, only `DataType CONFIG` and `STATE` have direct correlation and expression in YANG. `OPERATIONAL` is not supported.
- A single request cannot have both OpenConfig (OC) YANG and device YANG paths. A request must have only OC YANG paths or device YANG paths, but not both.
- `GetRequest` for root path (“/”: everything from **all** models) is not allowed.
- `GetRequest` for the top level of the device model (“/System”) is not allowed.
- gNMI Get returns all default values (ref. report-all mode in [RFC 6243](#) [4]).
- Subscribe supports the model `Cisco-NX-OS-syslog-oper`.
- Get does not support the model `Cisco-NX-OS-syslog-oper`.
- Query from the path `/system` does not return data from the path `/system/processes`. The specific path `/system/processes` should be used to query `openconfig-procmon` data.
- The following optional items are not supported:
  - Path prefix
  - Path alias
  - Wildcards in path
- A single `GetRequest` can have up to 10 paths.
- If the size of value field to be returned in `GetResponse` is over 12 MB, the system returns error status `grpc::RESOURCE_EXHAUSTED`.
- The maximum gRPC receive buffer size is set to 8 MB.

- The number of total concurrent sessions for Get is limited to five.
- Performing a Get operation when a large configuration is applied to the switch might cause the gRPC process to consume all available memory. If a memory exhaustion condition is hit, the following syslog is generated:

```
MTX-API: The memory usage is reaching the max memory resource limit (3072) MB
```

If this condition is hit several times consecutively, the following syslog is generated:

```
The process has become unstable and the feature should be restarted.
```

We recommend that you restart the gRPC feature at this point to continue normal processing of gNMI transactions.

# Set

## About Set

The Set RPC is used by a client to change the configuration of the device. The operations, which may be applied to the device data, are (in order) delete, replace, and update. All operations in a single Set request are treated as a transaction, meaning that all operations are successful or the device is rolled-back to the original state. The Set operations are applied in the order that is specified in the SetRequest. If a path is mentioned multiple times, the changes are applied even if they overwrite each other. The final state of the data is achieved with the final operation in the transaction. It is assumed that all paths specified in the SetRequest::delete, replace, update fields are CONFIG data paths and writable by the client.

For detailed information on the Set operation, refer to the Modifying State section of the gNMI Specification <https://github.com/openconfig/reference/blob/1cf43d2146f9ba70abb7f04f6b0f6eaa504cef05/rpc/gnmi/gnmi-specification.md>.

## Guidelines and Limitations for Set

The following are guidelines and limitations for Set:

- SetRequest.encoding supports only JSON.
- A single request cannot have both OpenConfig (OC) YANG and device YANG paths. A request must have only OC YANG paths or device YANG paths, but not both.
- Subscribe supports the model `Cisco-NX-OS-syslog-oper`.
- Query from the path `/system` does not return data from the path `/system/processes`. The specific path `/system/processes` should be used to query `openconfig-procmon` data.
- The following optional items are not supported:
  - Path prefix
  - Path alias
  - Wildcards in path
- A single SetRequest can have up to 20 paths.



- The maximum gRPC receive buffer size is set to 8 MB.
- The number of total concurrent sessions for Get is limited to five.
- Performing a Set operation when a large configuration is applied to the switch might cause the gRPC process to consume all available memory. If a memory exhaustion condition is hit, the following syslog is generated:

```
MTX-API: The memory usage is reaching the max memory resource limit (3072) MB
```

If this condition is hit several times consecutively, the following syslog is generated:

```
The process has become unstable and the feature should be restarted.
```

We recommend that you restart the gRPC feature at this point to continue normal processing of gNMI transactions.

- For the Set::Delete RPC, an MTX log message warns if the configuration being operated on may be too large:

```
Configuration size for this namespace exceeds operational limit. Feature may become unstable and require restart.
```

## Subscribe

### Guidelines and Limitations for Subscribe

Following are the guidelines and limitations for Subscribe:

- Beginning with Cisco NX-OS Release 9.3(3), Subscribe supports the OpenConfig model.
- The gNMI feature supports Subscribe and Capability as options of the gNMI service.
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12-MB maximum, the collected data is dropped.

You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.

- All paths within the same subscription request must have the same sample interval. If the same path requires different sample intervals, create multiple subscriptions.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The feature supports Cisco DME and Device YANG data models. Openconfig YANG is not supported.
- The gRPC process that supports gNMI uses the HIGH\_PRIO cgroup, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The commands are not XMLized in this release.
  - The gRPC agent retains gNMI calls for a maximum of 1 hour after the call has ended.

- If the total number of calls exceeds 2000, the gRPC agent purges ended calls based on an internal cleanup routine.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of 2 gRPC servers on each switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load might not be desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server processes requests independent of the other, and requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

## gNMI Payload

gNMI uses a specific payload format to subscribe to:

- DME Streams
- YANG Streams

Subscribe operations are supported with the following modes:

- ONCE: Subscribe and receive data once and close session.
- POLL: Subscribe and keep session open, client sends poll request each time data is needed.
- STREAM: Subscribe and receive data at specific cadence. The payload accepts values in nanoseconds  
1 second = 1000000000.
- ON\_CHANGE: Subscribe, receive a snapshot, and only receive data when something changes in the tree.

Setting modes:

- Each mode requires 2 settings, inside sub and outside sub
- ONCE: SAMPLE, ONCE
- POLL: SAMPLE, POLL
- STREAM: SAMPLE, STREAM
- ON\_CHANGE: ON\_CHANGE, STREAM

Origin

- DME: Subscribing to DME model

- device: Subscribing to YANG model

#### Name

- DME = subscribing to DME model
- Cisco-NX-OS-device = subscribing to YANG model

#### Encoding

- JSON = Stream will be send in JSON format.
- PROTO = Stream will be sent in protobuf.any format.

### Sample gNMI Payload for DME Stream



**Note** Different clients have their own input format.

```
{
 "SubscribeRequest":
 [
 {
 "_comment" : "ONCE request",
 "_delay" : 2,
 "subscribe":
 {
 "subscription":
 [
 {
 "_comment" : "1st subscription path",
 "path":
 {
 "origin": "DME",
 "elem":
 [
 {
 "name": "sys"
 },
 {
 "name": "bgp"
 }
]
 },
 "mode": "SAMPLE"
 }
],
 "mode": "ONCE",
 "allow_aggregation" : false,
 "use_models":
 [
 {
 "_comment" : "1st module",
 "name": "DME",
 "organization": "Cisco Systems, Inc.",
 "version": "1.0.0"
 }
],
 "encoding": "JSON"
 }
 }
]
}
```

```

 }
]
}

```

### Sample gNMI Payload YANG Stream

```

{
 "SubscribeRequest":
 [
 {
 "_comment" : "ONCE request",
 "_delay" : 2,
 "subscribe":
 {
 "subscription":
 [
 {
 "_comment" : "1st subscription path",
 "path":
 {
 "origin": "device",
 "elem":
 [
 {
 "name": "System"
 },
 {
 "name": "bgp-items"
 }
]
 },
 "mode": "SAMPLE"
 }
],
 "mode": "ONCE",
 "allow_aggregation" : false,
 "use_models":
 [
 {
 "_comment" : "1st module",
 "name": "Cisco-NX-OS-device",
 "organization": "Cisco Systems, Inc.",
 "version": "0.0.0"
 }
],
 "encoding": "JSON"
 }
 }
]
}

```

## Streaming Syslog

### About Streaming Syslog for gNMI

gRPC Network Management Interface (gNMI) defines a gRPC-based protocol for the modification and retrieval of configuration from a target device. Also, it defines the control and generation of telemetry streams from a target device to a data collection system. The intention is that a single gRPC service definition can

cover both configuration and telemetry - allowing a single implementation on the target, as well as a single NMS element to interact with the device via telemetry and configuration RPCs.

gNMI Subscribe is a new way of monitoring the network as it provides a real-time view of what's going on in your system by pushing the structured data as per gNMI Subscribe request.

Beginning with the Cisco NX-OS Release 9.3(3), support is added for gNMI Subscribe functionality.

#### gNMI Subscribe Support Detail

- Syslog-oper model streaming
  - stream\_on\_change

#### Telemetry Support Detail

- YANG model
  - Events only

This feature applies to Cisco Nexus 3500 platform switches with 8 GB or more of memory.

## Guidelines and Limitations for Streaming Syslog - gNMI

The following are guidelines and limitations for Streaming Syslog:

- An invalid syslog is not supported. For example, a syslog with a filter or query condition
- Only the following paths are supported:
  - Cisco-NX-OS-Syslog-oper:syslog
  - Cisco-NX-OS-Syslog-oper:syslog/messages
- The following modes are not supported:
  - Stream sample
  - POLL
- A request must be in the YANG model format.
- You can use the internal application or write your own application.
- The payload comes from the controller and gNMI sends a response.
- Encoding formats are JSON and PROTO.

## Syslog Native YANG Model

The YangModels are located [here](#).



---

**Note** The time-zone field is set only when the **clock format show-timezone syslog** is entered. By default, it's not set, therefore the time-zone field is empty.

---

```

PYANG Tree for Syslog Native Yang Model:
>>> pyang -f tree Cisco-NX-OS-infra-syslog-oper.yang
module: Cisco-NX-OS-syslog-oper
+--ro syslog
+--ro messages
+--ro message* [message-id]
+--ro message-id int32
+--ro node-name? string
+--ro time-stamp? uint64
+--ro time-of-day? string
+--ro time-zone? string
+--ro category? string
+--ro group? string
+--ro message-name? string
+--ro severity? System-message-severity
+--ro text? string

```

## Subscribe Request Example

The following is an example of a Subscribe request:

```

{
 "SubscribeRequest":
 [
 {
 "_comment" : "STREAM request",
 "_delay" : 2,
 "subscribe":
 {
 "subscription":
 [
 {
 "_comment" : "1st subscription path",
 "path":
 {
 "origin": "syslog-oper",
 "elem":
 [
 {
 "name": "syslog"
 },
 {
 "name": "messages"
 }
]
 },
 "mode": "ON_CHANGE"
 }
],
 "mode": "ON_CHANGE",
 "allow_aggregation" : false,
 "use_models":
 [
 {
 "_comment" : "1st module",
 "name": "Cisco-NX-OS-Syslog-oper",
 "organization": "Cisco Systems, Inc.",
 "version": "0.0.0"
 }
],
 "encoding": "JSON"
 }
 }
]
}

```

```

 }
]
}

```

## Sample PROTO Output

This is a sample of PROTO output.

```

#####

[Subscribe]-----

Reading from file ' /root/gnmi-console/testing_bl/stream_on_change/OC_SYSLOG.json '

Sat Aug 24 14:38:06 2019

Generating request : 1 -----

Comment : STREAM request

Delay : 2 sec(s) ...

Delay : 2 sec(s) DONE

subscribe {
 subscription {
 path {
 origin: "syslog-oper"

 elem {
 name: "syslog"
 }

 elem {
 name: "messages"
 }
 }

 mode: ON_CHANGE

 }

 use_models {
 name: "Cisco-NX-OS-Syslog-oper"
 organization: "Cisco Systems, Inc."
 version: "0.0.0"
 }

 encoding: PROTO
}

```

```
Thu Nov 21 14:26:41 2019
Received response 3 -----
update {
 timestamp: 1574375201665688000
 prefix {
 origin: "Syslog-oper"
 elem {
 name: "syslog"
 }
 elem {
 name: "messages"
 }
 }
 update {
 path {
 elem {
 name: "message-id"
 }
 }
 val {
 uint_val: 529
 }
 }
 update {
 path {
 elem {
 name: "node-name"
 }
 }
 val {
 string_val: "task-n9k-1"
 }
 }
 update {
 path {
 elem {
 name: "message-name"
 }
 }
 val {
 string_val: "VSHD_SYSLOG_CONFIG_I"
 }
 }
 update {
 path {
 elem {
 name: "text"
 }
 }
 val {
 string_val: "Configured from vty by admin on console0"
 }
 }
 update {
 path {
 elem {
 name: "group"
 }
 }
 val {
 string_val: "VSHD"
 }
 }
}
```



```

update {
 path {
 elem {
 name: "category"
 }
 }
 val {
 string_val: "VSHD"
 }
}
update {
 path {
 elem {
 name: "time-of-day"
 }
 }
 val {
 string_val: "Nov 21 2019 14:26:40"
 }
}
update {
 path {
 elem {
 name: "time-zone"
 }
 }
 val {
 string_val: ""
 }
}
update {
 path {
 elem {
 name: "time-stamp"
 }
 }
 val {
 uint_val: 1574375200000
 }
}
update {
 path {
 elem {
 name: "severity"
 }
 }
 val {
 uint_val: 5
 }
}

/Received -----
.

```

## Sample JSON Output

This is a sample JSON output.

```

[Subscribe]-----
Reading from file ' testing_b1/stream_on_change/OC_SYSLOG.json '

```

```

Tue Nov 26 11:47:00 2019
Generating request : 1 -----
Comment : STREAM request
Delay : 2 sec(s) ...
Delay : 2 sec(s) DONE
subscribe {
 subscription {
 path {
 origin: "syslog-oper"
 }
 elem {
 name: "syslog"
 }
 elem {
 name: "messages"
 }
 }
 mode: ON_CHANGE
}
use_models {
 name: "Cisco-NX-OS-Syslog-oper"
 organization: "Cisco Systems, Inc."
 version: "0.0.0"
}
}

Tue Nov 26 11:47:15 2019
Received response 5 -----
update {
 timestamp: 1574797636002053000
 prefix {
 }
 update {
 path {
 origin: "Syslog-oper"
 }
 elem {
 name: "syslog"
 }
 }
 val {
 json_val: "[{ \"messages\" : [[
 {\"message-id\":657},{\"node-name\": \"task-n9k-1\", \"time-stamp\": \"1574797635000\", \"time-of-day\": \"Nov
 26 2019
 11:47:15\", \"severity\":3, \"message-name\": \"HDR_L2LEN_ERR\", \"category\": \"ARP\", \"group\": \"ARP\", \"text\": \"arp
 [30318] Received packet with incorrect layer 2 address length (8 bytes), Normal pkt with
 S/D MAC: 003a.7d21.d55e ffff.ffff.ffff eff_ifc mgmt0(9), log_ifc mgmt0(9), phy_ifc
 mgmt0(9)\", \"time-zone\": \"\"}]] }]"
 }
}
}

/Received -----

```

## Troubleshooting

### Gathering TM-Trace Logs

1. tmtrace.bin -f gnmi-logs gnmi-events gnmi-errors following are available
2. Usage:

```

bash-4.3# tmtrace.bin -d gnmi-events | tail -30 Gives the last 30
}
}
}
[06/21/19 15:58:38.969 PDT f8f 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 124, sync_response:1
[06/21/19 15:58:43.210 PDT f90 3133] [3621780288][tm_ec_yang_data_processor.c:93] TM_EC:
[Y] Data received for 2799743488: 49
{
 "cdp-items" : {
 "inst-items" : {
 "if-items" : {
 "If-list" : [
 {
 "id" : "mgmt0",
 "ifstats-items" : {
 "v2Sent" : "74",
 "validV2Rcvd" : "79"
 }
 }
]
 }
 }
 }
}
[06/21/19 15:58:43.210 PDT f91 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 141, sync_response:1
[06/21/19 15:59:01.341 PDT f92 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/intf-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157935518, length: 3063619, sync_response:0
[06/21/19 15:59:03.933 PDT f93 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940881, length: 6756, sync_response:0
[06/21/19 15:59:03.940 PDT f94 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/lldp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940912, length: 8466, sync_response:1
bash-4.3#

```

## Gathering MTX-Internal Logs

1. Modify the following file with below /opt/mtx/conf/mtxlogger.cfg

```

<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">true</leaf>
 </container>
 <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>
 </container>

```

```

 <container name="fcn">
 <leaf name="enabled" type="boolean" default="true"></leaf>
 <leaf name="format" type="string"
default="$CLASS$::${FCNNAME}$($ARGS$)@${LINE$}"></leaf>
 </container>
 </container>
 <container name="facility">
 <leaf name="info" type="boolean" default="true">true</leaf>
 <leaf name="warning" type="boolean" default="true">true<
/leaf>
 <leaf name="error" type="boolean" default="true">true</leaf>
 <leaf name="debug" type="boolean" default="false">true<
/leaf>
 </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>
 <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>
 <leaf name="active" type="boolean" default="false"
>>false</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">MTX-EvtMgr</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">TM-ADPT</leaf>
 <leaf name="active" type="boolean" default="true"
>>false</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">TM-ADPT-JSON</leaf>
 <leaf name="active" type="boolean" default="true"
>>false</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">SYSTEM</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>

```

```

 <listitem>
 <leaf name="id" type="string">LIBUTILS</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">MTX-API</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">Model-*</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">Model-Cisco-NX-OS-
device</leaf>
 <leaf name="active" type="boolean" default="true"
>>false</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">Model-openconfig-bgp<
/leaf>
 <leaf name="active" type="boolean" default="true"
>>false</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">INST-MTX-API</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 <listitem>
 <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
 <leaf name="active" type="boolean" default="true"
>true</leaf>
 </listitem>
 </list>
</container>
</container>
</config>

```

2. Run "no feature grpc" / "feature grpc"

3. The /volataile directory houses the mtx-internal.log, the log rolls over over time so be sure to grab what you need before thenbash-4.3# cd /volatile/

```

bash-4.3# cd /volaiflcls -al
total 148
drwxrwxrwx 4 root root 340 Jun 21 15:47 .
drwxrwxr-t 64 root network-admin 1600 Jun 21 14:45 ..
-rw-rw-rw- 1 root root 103412 Jun 21 16:14 grpc-internal-log
-rw-r--r-- 1 root root 24 Jun 21 14:44 mtx-internal-19-06-21-14-46-21.log
-rw-r--r-- 1 root root 24 Jun 21 14:46 mtx-internal-19-06-21-14-46-46.log
-rw-r--r-- 1 root root 175 Jun 21 15:11 mtx-internal-19-06-21-15-11-57.log

```

```
-rw-r--r-- 1 root root 175 Jun 21 15:12 mtx-internal-19-06-21-15-12-28.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-17.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-42.log
-rw-r--r-- 1 root root 24 Jun 21 15:13 mtx-internal-19-06-21-15-14-22.log
-rw-r--r-- 1 root root 24 Jun 21 15:14 mtx-internal-19-06-21-15-19-05.log
-rw-r--r-- 1 root root 24 Jun 21 15:19 mtx-internal-19-06-21-15-47-09.log
-rw-r--r-- 1 root root 24 Jun 21 15:47 mtx-internal.log
-rw-rw-rw- 1 root root 355 Jun 21 14:44 netconf-internal-log
-rw-rw-rw- 1 root root 0 Jun 21 14:45 nginx_logflag
drwxrwxrwx 3 root root 60 Jun 21 14:45 uwsgipy
drwxrwxrwx 2 root root 40 Jun 21 14:43 virtual-instance
bash-4.3#.
```



## CHAPTER 24

# gNOI-gRPC Network Operations Interface

---

- [About gNOI, on page 293](#)
- [Supported gNOI RPCs, on page 293](#)
- [System Proto, on page 294](#)
- [OS Proto, on page 295](#)
- [Cert Proto, on page 296](#)
- [File Proto, on page 296](#)
- [gNOI Factory Reset, on page 297](#)
- [Guidelines and Limitations, on page 298](#)
- [Verifying gNOI, on page 298](#)

## About gNOI

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based micro-services for executing operational commands on network devices. The operational commands supported are Ping, Traceroute, Time, SwitchControlProcessor, Reboot, RebootStatus, CancelReboot, Activate and Verify.

gNOI uses gRPC as the transport protocol and the configuration is same as that of gNMI. For details on configuration, please refer to [Configuring gNMI](#).

To send gNOI RPC requests, user needs a client that implements the gNOI client interface for each RPC.

In Cisco NX-OS Release 10.1(1) the gNOI defines Remote Procedure Calls (RPCs) for a limited number of components and some of them related to hardware (like optical interfaces).

Proto files are defined for the gRPC micro-services and are available at [GitHub](#).

## Supported gNOI RPCs

The following are the supported gNOI RPCs:

Table 30:

Proto	gNOI RPC	Supported
System	Ping	Yes
	Traceroute	Yes
	Time	Yes
	SwitchControl Processor	Yes
	Reboot	Yes
	RebootStatus	Yes
	CancelReboot	Yes
OS	Activate	Yes
	Verify	Yes
Cert	LoadCertificate	Yes
File	Get	Yes
	Stat	Yes
	Remove	Yes

## System Proto

The System proto service is a collection of operational RPCs that allows the management of a target outside the configuration and telemetry pipeline.

The following are the RPC support details for System proto:

RPC	Support	Description	Limitation
Ping	ping/ping6 cli command	Executes the ping command on the target and streams back the results. Some targets may not stream any results until all results are available. If a packet count is not explicitly provided, 5 is used.	do_not_resolve option is not supported.



RPC	Support	Description	Limitation
Traceroute	traceroute/traceroute6 cli command	Executes the traceroute command on the target and streams back the results. Some targets may not stream any results until all results are available. Max hop count of 30 is used.	initial_ttl, marx_ttl, wait, do_not_fragment, do_not_resolve and l4protocol options are not supported.
Time	local time	Returns the current time on the target. Typically used to test if the target is responding.	-
SwitchControl Processor	system switchover cli command	Switches from the current route processor to the provided route processor. Switchover happens instantly and the response may not be guaranteed to return to the client.	Switchover occurs instantly. As a result, the response may not be guaranteed to return to the client.
Reboot	cli: reload [module]	Causes the target to reboot.	message option is not supported, delay option is supported for switch reload, and the path option accepts one module number.
RebootStatus	show version [module] cli command	Returns the status of the reboot for the target.	-
CancelReboot	reload cancel	Cancels any pending reboot request.	-



**Note** The SetPackage RPC is not supported.

## OS Proto

The OS service provides an interface for OS installation on a Target. The OS package file format is platform dependent. The platform must validate that the OS package that is supplied is valid and bootable. This must include a hash check against a known good hash. It is recommended that the hash is embedded in the OS package.

The Target manages its own persistent storage, and OS installation process. It stores a set of distinct OS packages, and always proactively frees up space for incoming new OS packages. It is guaranteed that the

Target always has enough space for a valid incoming OS package. The currently running OS packages must never be removed. The Client must expect that the last successfully installed package is available.

The following are the RPC support details for OS proto:

RPC	Support	Description	Limitation
Activate	install all nxos bootflash:///img_name	Sets the requested OS version as the version that is used at the next reboot. This RPC reboots the Target.	Cannot rollback or recover if the reboot fails.
Verify	show version	Verify checks the running OS version. This RPC may be called multiple times while the Target boots until it is successful.	-



**Note** The Install RPC is not supported.

## Cert Proto

The certificate management service is exported by targets. Rotate, Install and other Cert Proto RPCs are not supported.

The following are the RPC support details for Cert proto:

RPC	Support	Description	Limitation
LoadCertificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>	Loads a bundle of CA certificates.	-

## File Proto

The file proto streams messages based on the features of the file.proto RPCs. Put and other RPCs that are not listed here are not supported in File Proto.

Get, Stat, and Remove RPCs support file systems - bootflash, bootflash://sup-remote, logflash, logflash://sup-remote, usb, volatile, volatile://sup-remote and debug.

The following are the RPC support details for File proto:

RPC	Description	Limitation
Get	Get reads and streams the contents of a file from the target. The file is streamed by sequential messages, each containing up to 64 KB of data. A final message is sent prior to closing the stream that contains the hash of the data sent. An error is returned if the file does not exist or there was an error reading the file.	Maximum file size limit is 32 MB.
Stat	Stat returns metadata about a file on the target. An error is returned if the file does not exist or if there is an error in accessing the metadata.	-
Remove	Remove removes the specified file from the target. An error is returned if the file does not exist, is a directory, or the remove operation encounters an error.	-

## gNOI Factory Reset

The gNOI factory reset operation erases all persistent storage on the specified module. This includes configuration, all log data, and the full contents of flash and SSDs. The reset boots to the last boot image, erases all storage including license. gNOI factory reset supports two modes:

- A fast erase which can reformat and repartition only.
- A secure erase which can erase securely and wipe the data which is impossible to recover.

The gNOI factory reset operation as defined in `factory_reset.proto` erases all persistent storage on the device. Refer to `factory_reset.proto` link here [https://github.com/openconfig/gnoi/blob/master/factory\\_reset/factory\\_reset.proto](https://github.com/openconfig/gnoi/blob/master/factory_reset/factory_reset.proto).

The following is the example of a gNOI FactoryReset service:

```
/ The FactoryReset service exported by Targets.
service FactoryReset {
 // The Start RPC allows the Client to instruct the Target to immediately
 // clean all existing state and boot the Target in the same condition as it is
 // shipped from factory. State includes storage, configuration, logs,
 // certificates and licenses.
 //
 // Optionally allows rolling back the OS to the same version shipped from
 // factory.
 //
 // Optionally allows for the Target to zero-fill permanent storage where state
 // data is stored.
 //
```

```

// If any of the optional flags is set but not supported, a gRPC Status with
// code INVALID_ARGUMENT must be returned with the details value set to a
// properly populated ResetError message.
rpc Start(StartRequest) returns (StartResponse);
}

message StartRequest {
 // Instructs the Target to rollback the OS to the same version as it shipped
 // from factory.
 bool factory_os = 1;
 // Instructs the Target to zero fill persistent storage state data.
 bool zero_fill = 2;
}

```

The following are the details of the arguments used in gNOI Factory Reset:

- **factory\_os = false**: Specifies to rollback to the OS version as shipped from factory. Setting to **true** on NX-OS is not supported, and it is mandatory to preserve the current boot image.
- **zero\_fill**: Specifies whether to perform more time consuming and comprehensive secure erase.
  - **zero\_fill = true**: Specifies factory-reset module all preserve-image force.
  - **zero\_fill = false**: Specifies factory-reset module all bypass-secure-erase preserve-image force.

## Guidelines and Limitations

The gNOI feature has the following guidelines and limitations:

- A maximum of 16 active gNOI RPCs are supported.
- The Cisco Nexus 9000 series switches would run one endpoint with one gNMI service and two gNOI microservices.
- In 10.1(1) release, the gNOI RPCs are implemented with the equivalent CLI. The existing CLI restrictions or valid options remain as applicable.
- Beginning with 10.2(1)F release, the file.proto and cert.proto RPCs are supported.
- gRPC traffic destined for a Nexus device will hit the control-plane policer (CoPP) in the default class. To limit the possibility of gRPC drops, configure a custom CoPP policy using the gRPC configured port in the management class.

## Verifying gNOI

To verify the gNOI configuration, enter the following commands:

Command	Description
clear grpc gnoi rpc	Serves to clean up the counters or calls.
debug grpc events {events errors}	Debugs the events and errors from the event history.
show grpc nxsdk event-history {events errors}	

Command	Description
show grpc internal gnoi rpc {summary detail}	An internal keyword command added for serviceability.





## CHAPTER 25

# Model Driven Telemetry

- [About Telemetry, on page 301](#)
- [Licensing Requirements for Telemetry, on page 303](#)
- [Installing and Upgrading Telemetry, on page 303](#)
- [Guidelines and Limitations for Model Driven Telemetry, on page 304](#)
- [Configuring Telemetry Using the CLI, on page 311](#)
- [Configuring Telemetry Using the NX-API, on page 332](#)
- [Telemetry Path Labels, on page 346](#)
- [Native Data Source Paths, on page 363](#)
- [Streaming Syslog, on page 375](#)
- [Additional References, on page 382](#)

## 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.)

Starting with Cisco NX-OS Release 9.2(1), telemetry now supports streaming to IPv6 destinations and IPv4 destinations.

Starting with Cisco NX-OS Release 7.0(3)I7(1), 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.

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 is supported with the following behaviors:

- **System Reload** — During a system reload, any telemetry configuration and streaming services are restored.
- **Process Restart** — If the telemetry process freezes or restarts for any reason, configuration, and streaming services are restored when telemetry is restarted.

## Licensing Requirements for Telemetry

Product	License Requirement
Cisco NX-OS	Telemetry requires no license. Any feature not included in a license package is bundled with the 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 located in the `/rpms` directory and is named as follows:

**telemetry-version-build\_ID.libn32\_n9000.rpm**

**telemetry-version-build\_ID.libn32\_n3000.rpm**

As in the following example:

```
telemetry-2.0.0-7.0.3.I5.1.lib32_n9000.rpm
```

```
telemetry-2.0.0-7.0.3.I5.1.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:

```
dnf upgrade telemetry_new_version.rpm
```

The application is upgraded and the change appears when the application is started again.

### Downgrading to a Previous Version

To downgrade the telemetry application to a previous version, use the following command from the `bash` prompt:

```
dnf 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 will only show the active installed RPM after an upgrade has occurred. The default RPM that comes bundled with the NX-OS will not be displayed.

---

## Guidelines and Limitations for Model Driven Telemetry

Telemetry has the following configuration guidelines and limitations:

- Cisco NX-OS releases that support the data management engine (DME) Native Model support Telemetry.
- Support is in place for the following:
  - DME data collection
  - NX-API data sources
  - Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport
  - 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 any cadences below the minimum value may result in undesirable system behavior.
- Telemetry supports up to five remote management receivers (destinations). Configuring more than five remote receivers may result in undesirable system behavior.
- 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.
- The following are guidelines for setting the telemetry cadence on YANG paths:
  - YANG streaming collection takes one thread. If multiple YANG paths exist in telemetry, each must run on a different cadence to prevent simultaneous scheduling and any resulting delay.
  - Before configuring the telemetry cadence for YANG paths, determine the total streaming time and configure the cadence to a value greater than the total streaming time. See *Configuring Cadence for YANG Path*.

### 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. When downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up. This sequence avoids the 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(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
```

```

switch(conf-tm-sensor)# subscription 600`
switch(conf-tm-sub)# dst-grp 100`
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#

```

### gRPC Error Behavior

The switch client disables the connection to the gRPC receiver if the gRPC receiver sends 20 errors. 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

Starting with Release 9.2(1), support for gRPC chunking has been added. 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.

The gRPC user must do the gRPC chunking. The gRPC client side does the fragmentation, and the gRPC server side does the reassembly. 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*.

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, such as the Cisco MDS series switches:

```
MDS-9706-86(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 only the summary or counter is needed, then avoid dumping a whole show command output to 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 or 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"
 }
 }
 }
]
 }
}
```

## Certificate Trustpoint Support

Beginning in NX-OS release 10.1(1), the **trustpoint** keyword is added in the existing global level command.

The following is the command syntax:

```
switch(config-telemetry)# certificate ?
trustpoint specify trustpoint label
WORD .pem certificate filename (Max Size 256)
switch(config-telemetry)# certificate trustpoint
WORD trustpoint label name (Max Size 256)
switch(config-telemetry)# certificate trustpoint trustpoint1 ?
WORD Hostname associated with certificate (Max Size 256)
switch(config-telemetry)#certificate trustpoint trustpoint1 foo.test.google.fr
```

## Destination Hostname Support

Beginning in NX-OS release 10.1(1), the **host** keyword is added in destination-group command.

The following is the example for the destination hostname support:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ?
certificate Specify certificate
host Specify destination host
ip Set destination IPv4 address
ipv6 Set destination IPv6 address
...
switch(conf-tm-dest)# host ?
```

```

A.B.C.D|A:B::C:D|WORD IPv4 or IPv6 address or DNS name of destination
switch(conf-tm-dest)#

switch(conf-tm-dest)# host abc port 11111 ?
protocol Set transport protocol
switch(conf-tm-dest)# host abc port 11111 protocol ?
HTTP
UDP
gRPC
switch(conf-tm-dest)# host abc port 11111 protocol gRPC ?
encoding Set encoding format
switch(conf-tm-dest)# host abc port 11111 protocol gRPC encoding ?
Form-data Set encoding to Form-data only
GPB Set encoding to GPB only
GPB-compact Set encoding to Compact-GPB only
JSON Set encoding to JSON
XML Set encoding to XML
switch(conf-tm-dest)# host ip address 1.1.1.1 port 2222 protocol HTTP encoding JSON
<CR>

```

### Support for Node ID

Beginning in NX-OS release 10.1(1), you can configure a custom Node ID string for a telemetry receiver through the **use-nodeid** command. By default, the host name is used, but support for a node ID enables you to set or change the identifier for the `node_id_str` of the telemetry receiver data.

You can assign the node ID through the telemetry destination profile, by using the **usenode-id** command. This command is optional.

The following example shows configuring the node ID.

```

switch-1(config)# telemetry
switch-1(config-telemetry)# destination-profile
switch-1(conf-tm-dest-profile)# use-nodeid test-srvr-10
switch-1(conf-tm-dest-profile)#

```

The following example shows a telemetry notification on the receiver after the node ID is configured.

```

Telemetry receiver:
=====
node_id_str: "test-srvr-10"
subscription_id_str: "1"
encoding_path: "sys/ch/psuslot-1/psu"
collection_id: 3896
msg_timestamp: 1559669946501

```

Use the **use-nodeid** sub-command under the **host** command. The destination level **use-nodeid** configuration precedes the global level configuration.

The following example shows the command syntax:

```

switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# use-nodeid ?
WORD Node ID (Max Size 128)
switch(conf-tm-dest-host)# use-nodeid session_1:18112

```

The following example shows the output from the Telemetry receiver:

```

>> Message size 923
Telemetry msg received @ 23:41:38 UTC
 Msg Size: 11
 node_id_str : session_1:18112
 collection_id : 3118
 data_source : DME

```



```

encoding_path : sys/ch/psuslot-1/psu
collection_start_time : 1598485314721
collection_end_time : 1598485314721
data :

```

### Support for Streaming of YANG Models

Starting with Release 9.2(1), telemetry supports the YANG ("Yet Another Next Generation") data modeling language. Telemetry supports data streaming for both device YANG and OpenConfig YANG.

For more information on the YANG data modeling language, see *Infrastructure Overview* and *RESTConf Agent*.

## 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.

#### Before you begin

Your switch must be running Cisco NX-OS Release 7.3(0)I5(1) or a later release.

#### SUMMARY STEPS

1. (Optional) **openssl** *argument*
2. **configure terminal**
3. **feature telemetry**
4. **feature nxapi**
5. **nxapi use-vrf management**
6. **telemetry**
7. (Optional) **certificate** *certificate\_path host\_URL*
8. (Optional) Specify a transport VRF or enable telemetry compression for gRPC transport.
9. **sensor-group** *sgrp\_id*
10. (Optional) **data-source** *data-source-type*
11. **path** *sensor\_path depth 0 [filter-condition filter] [alias path\_alias]*
12. **destination-group** *dgrp\_id*
13. (Optional) **ip address** *ip\_address port port protocol procedural-protocol encoding encoding-protocol*
14. (Optional) **ipv6 address** *ipv6\_address port port protocol procedural-protocol encoding encoding-protocol*
15. **ip\_version address** *ip\_address port portnum*
16. (Optional) **use-chunking size** *chunking\_size*
17. **subscription** *sub\_id*
18. **snsr-grp** *sgrp\_id sample-interval interval*
19. **dst-grp** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<p>(Optional) <code>openssl argument</code></p> <p><b>Example:</b> Generate an SSL/TLS certificate using a specific argument, such as the following:</p> <ul style="list-style-type: none"> <li>To generate a private RSA key: <code>openssl genrsa -cipher -out filename.key cipher-bit-length</code> For example:  <pre>switch# openssl genrsa -des3 -out server.key 2048</pre> </li> <li>To write the RSA key: <code>openssl rsa -in filename.key -out filename.key</code> For example:  <pre>switch# openssl rsa -in server.key -out server.key</pre> </li> <li>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> For example:  <pre>switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'</pre> </li> <li>To create a public key: <code>openssl x509 -req -encoding-standard -days timeframe -in filename.csr -signkey filename.key -out filename.csr</code> For example:  <pre>switch# openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt</pre> </li> </ul>	Create an SSL or TLS certificate on the server that receives the data, where the <code>private.key</code> file is the private key and the <code>public.crt</code> is the public key.
<b>Step 2</b>	<p><b>configure terminal</b></p> <p><b>Example:</b>  <pre>switch# configure terminal switch(config)#</pre> </p>	Enter the global configuration mode.
<b>Step 3</b>	<b>feature telemetry</b>	Enable the streaming telemetry feature.
<b>Step 4</b>	<b>feature nxapi</b>	Enable NX-API.

	Command or Action	Purpose
Step 5	<b>nxapi use-vrf management</b>	Enable the VRF management to be used for NX-API communication.
Step 6	<b>telemetry</b> <b>Example:</b> switch(config)# <b>telemetry</b> switch(config-telemetry)#	Enter configuration mode for streaming telemetry.
Step 7	(Optional) <b>certificate</b> <i>certificate_path host_URL</i> <b>Example:</b> switch(config-telemetry)# <b>certificate</b> /bootflash/server.key localhost	Use an existing SSL/TLS certificate.
Step 8	(Optional) Specify a transport VRF or enable telemetry compression for gRPC transport. <b>Example:</b> switch(config-telemetry)# <b>destination-profile</b> switch(conf-tm-dest-profile)# <b>use-vrf default</b> switch(conf-tm-dest-profile)# <b>use-compression gzip</b> switch(conf-tm-dest-profile)# <b>use-retry size 10</b> switch(conf-tm-dest-profile)# <b>source-interface loopback1</b>	<ul style="list-style-type: none"> <li>• Enter the <b>destination-profile</b> command to specify the default destination profile.</li> <li>• Enter any of the following commands: <ul style="list-style-type: none"> <li>• <b>use-vrf</b> <i>vrf</i> to specify the destination VRF.</li> <li>• <b>use-compression gzip</b> to specify the destination compression method.</li> <li>• <b>use-retry size</b> <i>size</i> to specify the send retry details, with a retry buffer size between 10–1500 megabytes.</li> <li>• <b>source-interface</b> <i>interface-name</i> to stream data from the configured interface to a destination with the source IP address.</li> </ul> </li> </ul> <p><b>Note</b> After configuring the <b>use-vrf</b> 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 action ensures that the telemetry data streams to the same destination IP address in the new VRF.</p>
Step 9	<b>sensor-group</b> <i>srgp_id</i> <b>Example:</b> switch(config-telemetry)# <b>sensor-group 100</b> switch(conf-tm-sensor)#	<p>Create a sensor group with ID <i>srgp_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	(Optional) <b>data-source</b> <i>data-source-type</i> <b>Example:</b> switch(config-telemetry)# <b>data-source NX-API</b>	<p>Select a data source. Select from either YANG, DME or NX-API as the data source.</p> <p><b>Note</b> DME is the default data source.</p>

	Command or Action	Purpose
<b>Step 11</b>	<p><b>path</b> <i>sensor_path</i> <b>depth 0</b> [<b>filter-condition</b> <i>filter</i>] [<b>alias</b> <i>path_alias</i>]</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>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(l2Bd.operSt, "down")</pre> <p>Use the following syntax for state-based filtering to trigger only when <b>operSt</b> changes from <b>up</b> to <b>down</b>, 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(l2Bd.operSt),eq(l2Bd.operSt,"down"))</pre> <p>Use the following syntax to distinguish the path on the UTR side.</p> <pre>switch(conf-tm-sensor)# path sys/ch/ftslot-1/ft alias ft_1</pre> </li> <li>The following command is applicable for NX-API, not for DME or YANG: <pre>switch(conf-tm-sensor)# path "show interface" depth 0</pre> </li> <li>The following command is applicable for device YANG: <pre>switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items</pre> </li> <li>The following commands are applicable for OpenConfig YANG: <pre>switch(conf-tm-sensor)# path openconfig-bgp:bgp  switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items alias bgp_alias</pre> </li> <li>The following command is applicable for NX-API: <pre>switch(conf-tm-sensor)# path "show interface" depth 0 alias sh_int_alias</pre> </li> <li>The following command is applicable for OpenConfig: <pre>switch(conf-tm-sensor)# path openconfig-bgp:bgp alias oc_bgp_alias</pre> </li> </ul>	<p>Add a sensor path to the sensor group.</p> <ul style="list-style-type: none"> <li>Beginning with the Cisco NX-OS 9.3(5) release, the <b>alias</b> keyword is introduced.</li> <li>The <b>depth</b> setting specifies the retrieval level for the sensor path. Depth settings of <b>0 - 32</b>, <b>unbounded</b> are supported. <p><b>Note</b> <b>depth 0</b> is the default depth.</p> <p>NX-API-based sensor paths can only use <b>depth 0</b>.</p> <p>If a path is subscribed for the event collection, the depth only supports 0 and unbounded. Other values would be treated as 0.</p> </li> <li>The optional <b>filter-condition</b> 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 <b>sys/bd/bd-[vlan]</b> of <b>eq(l2Bd.operSt, "down")</b> triggers when the operSt changes, and when the DN's property changes while the operSt remains <b>down</b>, such as a <b>no shutdown</b> command is issued while the VLAN is operationally <b>down</b>.</p> <p><b>Note</b> 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> </li> <li>For the YANG model, the sensor path format is as follows: <i>module_name: YANG_path</i>, where <i>module_name</i> is the name of the YANG model file. For example: <ul style="list-style-type: none"> <li>For device YANG: <pre>Cisco-NX-OS-device:System/bgp-items/inst-items</pre> </li> <li>For OpenConfig YANG: <pre>openconfig-bgp:bgp</pre> </li> </ul> </li> </ul> <p><b>Note</b> The <b>depth</b>, <b>filter-condition</b>, and <b>query-condition</b> parameters are not supported for YANG currently.</p>

	Command or Action	Purpose
		<p>For the openconfig YANG models, go to <a href="https://github.com/YangModels/yang/tree/master/vendor/cisco/nx">https://github.com/YangModels/yang/tree/master/vendor/cisco/nx</a> and navigate to the appropriate folder for the latest release.</p> <p>Instead of installing a specific model, you can install the openconfig-all RPM which has all the OpenConfig models. See <i>Adding Patch RPMs from Bash</i> topic for more information on installing patch RPMs.</p> <p>For example:</p> <pre><b>install add mtx-openconfig-bgp-1.0.0.0-7.0.3.IHD8.1.lib32_n9000.rpm activate</b></pre>
<b>Step 12</b>	<p><b>destination-group</b> <i>dgrp_id</i></p> <p><b>Example:</b></p> <pre>switch(conf-tm-sensor) # <b>destination-group</b> 100 switch(conf-tm-dest) #</pre>	<p>Create a destination group and enter destination group configuration mode.</p> <p>Currently <i>dgrp_id</i> only supports numeric ID values.</p>
<b>Step 13</b>	<p>(Optional) <b>ip address</b> <i>ip_address</i> <b>port</b> <i>port</i> <b>protocol</b> <i>procedural-protocol</i> <b>encoding</b> <i>encoding-protocol</i></p> <p><b>Example:</b></p> <pre>switch(conf-tm-sensor) # <b>ip address</b> 171.70.55.69 <b>port</b> 50001 <b>protocol</b> gRPC <b>encoding</b> GPB switch(conf-tm-sensor) # <b>ip address</b> 171.70.55.69 <b>port</b> 50007 <b>protocol</b> HTTP <b>encoding</b> JSON  switch(conf-tm-sensor) # <b>ip address</b> 171.70.55.69 <b>port</b> 50009 <b>protocol</b> UDP <b>encoding</b> JSON</pre>	<p>Specify an IPv4 IP address and port to receive encoded telemetry data.</p> <p><b>Note</b> gRPC is the default transport protocol. GPB is the default encoding.</p>
<b>Step 14</b>	<p>(Optional) <b>ipv6 address</b> <i>ipv6_address</i> <b>port</b> <i>port</i> <b>protocol</b> <i>procedural-protocol</i> <b>encoding</b> <i>encoding-protocol</i></p> <p><b>Example:</b></p> <pre>switch(conf-tm-sensor) # <b>ipv6 address</b> 10:10::1 <b>port</b> 8000 <b>protocol</b> gRPC <b>encoding</b> GPB switch(conf-tm-sensor) # <b>ipv6 address</b> 10:10::1 <b>port</b> 8001 <b>protocol</b> HTTP <b>encoding</b> JSON switch(conf-tm-sensor) # <b>ipv6 address</b> 10:10::1 <b>port</b> 8002 <b>protocol</b> UDP <b>encoding</b> JSON</pre>	<p>Specify an IPv6 IP address and port to receive encoded telemetry data.</p> <p><b>Note</b> gRPC is the default transport protocol. GPB is the default encoding.</p>
<b>Step 15</b>	<p><b>ip_version</b> <b>address</b> <i>ip_address</i> <b>port</b> <i>portnum</i></p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>For IPv4: <pre>switch(conf-tm-dest) # <b>ip address</b> 1.2.3.4 <b>port</b> 50003</pre> </li> <li>For IPv6:</li> </ul>	<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 is specified by this profile.</p>

	Command or Action	Purpose
	<pre>switch(conf-tm-dest)# ipv6 address 10:10::1 port 8000</pre>	
<b>Step 16</b>	(Optional) <b>use-chunking size</b> <i>chunking_size</i> <b>Example:</b> <pre>switch(conf-tm-dest)# use-chunking size 64</pre>	Enable gRPC chunking and set the chunking size, between 64-4096 bytes. See the section "Support for gRPC Chunking" for more information.
<b>Step 17</b>	<b>subscription</b> <i>sub_id</i> <b>Example:</b> <pre>switch(conf-tm-dest)# subscription 100 switch(conf-tm-sub)#</pre>	Create a subscription node with ID and enter the subscription configuration mode.  Currently <i>sub_id</i> only supports numeric ID values.  <b>Note</b> When subscribing to a DN, check whether the DN is supported by DME using REST to ensure that events will stream.
<b>Step 18</b>	<b>snsr-grp</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i> <b>Example:</b> <pre>switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000</pre>	Link the sensor group with ID <i>sgrp_id</i> to this subscription and set the data sampling interval 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.
<b>Step 19</b>	<b>dst-grp</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch(conf-tm-sub)# dst-grp 100</pre>	Link the destination group with ID <i>dgrp_id</i> to this subscription.

## Configuring Cadence for YANG Paths

The cadence for YANG paths must be greater than the total streaming time. If the total streaming time and cadence are incorrectly configured, gathering telemetry data can take longer than the streaming interval. In this situation, you can see:

- Queues that incrementally fill because telemetry data is accumulating faster than it is streaming to the receiver.
- Stale telemetry data which is not from the current interval.

Configure the cadence to a value greater than the total streaming time.

### SUMMARY STEPS

1. **show telemetry control database sensor-groups**
2. **sensor group** *number*

3. **subscription** *number*
4. **snsr-grp** *number sample-interval milliseconds*
5. **show system resources**

## DETAILED STEPS

### Procedure

	Command or Action	Purpose																																																																																																																														
Step 1	<p><b>show telemetry control database sensor-groups</b></p> <p><b>Example:</b></p> <pre>switch-1# show telemetry control database sensor-groups Sensor Group Database size = 2</pre> <table border="1"> <thead> <tr> <th>Row ID</th> <th>Sensor Group ID</th> <th>Sensor Group type</th> <th>Sampling interval(ms)</th> <th>Linked subscriptions</th> <th>SubID</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>2</td> <td>Timer</td> <td>/YANG</td> <td>5000</td> <td></td> </tr> <tr> <td></td> <td>/Running</td> <td>1</td> <td></td> <td>1</td> <td></td> </tr> <tr> <td colspan="6">Collection Time in ms (Cur/Min/Max): 2444/2294/2460</td> </tr> <tr> <td colspan="6">Encoding Time in ms (Cur/Min/Max): 56/55/57</td> </tr> <tr> <td colspan="6">Transport Time in ms (Cur/Min/Max): 0/0/1</td> </tr> <tr> <td colspan="6"><b>Streaming Time in ms (Cur/Min/Max): 2515/2356/28403</b></td> </tr> <tr> <td colspan="6">Collection Statistics:</td> </tr> <tr> <td colspan="6">collection_id_dropped = 0</td> </tr> <tr> <td colspan="6">last_collection_id_dropped = 0</td> </tr> <tr> <td colspan="6">drop_count = 0</td> </tr> <tr> <td>2</td> <td>1</td> <td>Timer</td> <td>/YANG</td> <td>5000</td> <td></td> </tr> <tr> <td></td> <td>/Running</td> <td>1</td> <td></td> <td>1</td> <td></td> </tr> <tr> <td colspan="6">Collection Time in ms (Cur/Min/Max): 144/142/1471</td> </tr> <tr> <td colspan="6">Encoding Time in ms (Cur/Min/Max): 0/0/1</td> </tr> <tr> <td colspan="6">Transport Time in ms (Cur/Min/Max): 0/0/0</td> </tr> <tr> <td colspan="6"><b>Streaming Time in ms (Cur/Min/Max): 149/147/23548</b></td> </tr> <tr> <td colspan="6">Collection Statistics:</td> </tr> <tr> <td colspan="6">collection_id_dropped = 0</td> </tr> <tr> <td colspan="6">last_collection_id_dropped = 0</td> </tr> <tr> <td colspan="6">drop_count = 0</td> </tr> </tbody> </table> <pre>switch-1# telemetry  destination-group 1   ip address 192.0.2.1 port 9000 protocol HTTP   encoding JSON   sensor-group 1    data-source YANG    path /Cisco-NX-OS-device:System/procsys-items   depth unbounded   sensor-group 2    data-source YANG    path /Cisco-NX-OS-device:System/intf-items/phys-items   depth unbounded</pre>	Row ID	Sensor Group ID	Sensor Group type	Sampling interval(ms)	Linked subscriptions	SubID	1	2	Timer	/YANG	5000			/Running	1		1		Collection Time in ms (Cur/Min/Max): 2444/2294/2460						Encoding Time in ms (Cur/Min/Max): 56/55/57						Transport Time in ms (Cur/Min/Max): 0/0/1						<b>Streaming Time in ms (Cur/Min/Max): 2515/2356/28403</b>						Collection Statistics:						collection_id_dropped = 0						last_collection_id_dropped = 0						drop_count = 0						2	1	Timer	/YANG	5000			/Running	1		1		Collection Time in ms (Cur/Min/Max): 144/142/1471						Encoding Time in ms (Cur/Min/Max): 0/0/1						Transport Time in ms (Cur/Min/Max): 0/0/0						<b>Streaming Time in ms (Cur/Min/Max): 149/147/23548</b>						Collection Statistics:						collection_id_dropped = 0						last_collection_id_dropped = 0						drop_count = 0						<p>Calculate the total streaming time.</p> <p>The total streaming time is the sum of the individual current streaming times of each sensor group. Individual streaming times are displayed in Streaming time in ms (Cur). In this example, total streaming time is 2.664 seconds (2515 milliseconds plus 149 milliseconds).</p> <p>Compare the configured cadence to the total streaming time for the sensor group.</p> <p>The cadence is displayed in sample-interval. In this example, the cadence is correctly configured because the total streaming time (2.664 seconds) is less than the cadence (5.000 seconds, which is the default).</p>
Row ID	Sensor Group ID	Sensor Group type	Sampling interval(ms)	Linked subscriptions	SubID																																																																																																																											
1	2	Timer	/YANG	5000																																																																																																																												
	/Running	1		1																																																																																																																												
Collection Time in ms (Cur/Min/Max): 2444/2294/2460																																																																																																																																
Encoding Time in ms (Cur/Min/Max): 56/55/57																																																																																																																																
Transport Time in ms (Cur/Min/Max): 0/0/1																																																																																																																																
<b>Streaming Time in ms (Cur/Min/Max): 2515/2356/28403</b>																																																																																																																																
Collection Statistics:																																																																																																																																
collection_id_dropped = 0																																																																																																																																
last_collection_id_dropped = 0																																																																																																																																
drop_count = 0																																																																																																																																
2	1	Timer	/YANG	5000																																																																																																																												
	/Running	1		1																																																																																																																												
Collection Time in ms (Cur/Min/Max): 144/142/1471																																																																																																																																
Encoding Time in ms (Cur/Min/Max): 0/0/1																																																																																																																																
Transport Time in ms (Cur/Min/Max): 0/0/0																																																																																																																																
<b>Streaming Time in ms (Cur/Min/Max): 149/147/23548</b>																																																																																																																																
Collection Statistics:																																																																																																																																
collection_id_dropped = 0																																																																																																																																
last_collection_id_dropped = 0																																																																																																																																
drop_count = 0																																																																																																																																

	Command or Action	Purpose
	<pre>subscription 1   dst-grp 1   snsr-grp 1 sample-interval 5000   snsr-grp 2 sample-interval 5000</pre>	
<b>Step 2</b>	<p><b>sensor group</b> <i>number</i></p> <p><b>Example:</b></p> <pre>switch-1(config-telemetry)# sensor group1</pre>	If the total streaming time is not less than the cadence, enter the sensor group for which you want to set the interval.
<b>Step 3</b>	<p><b>subscription</b> <i>number</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sensor)# subscription 100</pre>	Edit the subscription for the sensor group.
<b>Step 4</b>	<p><b>snsr-grp</b> <i>number</i> <b>sample-interval</b> <i>milliseconds</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# snsr-grp number sample-interval 5000</pre>	<p>For the appropriate sensor group, set the sample interval to a value greater than the total streaming time.</p> <p>In this example, the sample interval is set to 5.000 seconds, which is valid because it is larger than the total streaming time of 2.664 seconds.</p>
<b>Step 5</b>	<p><b>show system resources</b></p> <p><b>Example:</b></p> <pre>switch-1# show system resources Load average:  1 minute: 0.38   5 minutes: 0.43                15 minutes: 0.43 Processes:    555 total, 3 running CPU states   :  24.17% user,   4.32% kernel,                71.50% idle                CPU0 states:   0.00% user,   2.12% kernel,                97.87% idle                CPU1 states:  86.00% user,  11.00% kernel,                3.00% idle                CPU2 states:   8.08% user,   3.03% kernel,                88.88% idle                CPU3 states:   0.00% user,   1.02% kernel,                98.97% idle Memory usage: 16400084K total,  5861652K used,                10538432K free Current memory status: OK</pre>	<p>Check the CPU usage.</p> <p>If the CPU user state shows high usage, as shown in this example, your cadence and streaming value are not configured correctly. Repeat this procedure to properly configure the cadence.</p>

## Configuration Examples for Telemetry Using the CLI

The following steps describe how to configure a single telemetry DME stream with a ten second cadence with GPB encoding.

```
switch# configure terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(config-tm-dest)# ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
switch(config-tm-dest)# exit
switch(config-telemetry)# sensor group sg1
switch(config-tm-sensor)# data-source DME
```



```
switch(config-tm-dest)# path interface depth unbounded query-condition keep-data-type
switch(config-tm-dest)# subscription 1
switch(config-tm-dest)# dst-grp 1
switch(config-tm-dest)# snsr grp 1 sample interval 10000
```

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, and 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(config-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

```

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

```

switch(config) # telemetry
switch(config-telemetry) # sensor-group 100
switch(conf-tm-sensor) # path 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, and a subscription can be linked to multiple sensor groups and destination groups, as shown in this example.

```

switch(config) # telemetry
switch(config-telemetry) # sensor-group 100
switch(conf-tm-sensor) # path 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
!Time: Thu Oct 13 21:10:12 2016

version 7.0(3)I5(1)
feature telemetry

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

```

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 yang direct-path cisco-nxos-device

This command displays YANG paths that are directly encoded to perform better than other paths.

```

switch# show telemetry yang direct-path cisco-nxos-device
) Cisco-NX-OS-device:System/lldp-items
2) Cisco-NX-OS-device:System/acl-items
3) Cisco-NX-OS-device:System/mac-items
4) Cisco-NX-OS-device:System/intf-items
5) Cisco-NX-OS-device:System/procsys-items/sysload-items
6) Cisco-NX-OS-device:System/ospf-items
7) Cisco-NX-OS-device:System/procsys-items
8) Cisco-NX-OS-device:System/ipqos-items/queuing-items/policy-items/out-items
9) Cisco-NX-OS-device:System/mac-items/static-items
10) Cisco-NX-OS-device:System/ch-items
11) Cisco-NX-OS-device:System/cdp-items
12) Cisco-NX-OS-device:System/bd-items
13) Cisco-NX-OS-device:System/eps-items
14) Cisco-NX-OS-device:System/ipv6-items

```

### 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 database sensor-paths

This command displays sensor path details for telemetry configuration, including counters for encoding, collection, transport, and streaming.

```
switch-1(conf-tm-sub)# show telemetry control database sensor-paths
```

```
Sensor Path Database size = 4
```

```

Row ID Subscribed Linked Groups Sec Groups Retrieve level Path(GroupId) : Query :
Filter

```

```
1 No 1 0 Full sys/cdp(1) : NA : NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 65785/65785/65785
Collection Time in ms (Cur/Min/Max): 10/10/55
Encoding Time in ms (Cur/Min/Max): 8/8/9
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 18/18/65
```

```
2 No 1 0 Self show module(2) : NA : NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 1107/1106/1107
Collection Time in ms (Cur/Min/Max): 603/603/802
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/1
Streaming Time in ms (Cur/Min/Max): 605/605/803
```

```
3 No 1 0 Full sys/bgp(1) : NA : NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 0/0/44
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1/1/44
```

```
4 No 1 0 Self show version(2) : NA : NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 2442/2441/2442
```

```
Collection Time in ms (Cur/Min/Max): 1703/1703/1903
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1703/1703/1904
```

```
switch-1(conf-tm-sub)#
```

### 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
```

```
 Memory:
```

```
 Allowed Memory Limit = 1181116006 bytes
 Occupied Memory = 93265920 bytes
```

```
Queue Statistics:
```

```
 Request Queue:
```

```
 High Priority Queue:
```

```
 Info:
```

```
 Actual Size = 50 Current Size = 0
 Max Size = 0 Full Count = 0
```

```
 Errors:
```

```
 Enqueue Error = 0 Dequeue Error = 0
```

```
 Low Priority Queue:
```

```
 Info:
```

```
 Actual Size = 50 Current Size = 0
 Max Size = 0 Full Count = 0
```

```
 Errors:
```

```
 Enqueue Error = 0 Dequeue Error = 0
```

```
 Data Queue:
```

```
 High Priority Queue:
```

```
 Info:
```

```
 Actual Size = 50 Current Size = 0
```



```

 Max Size = 0 Full Count = 0
Errors:
 Enqueue Error = 0 Dequeue Error = 0
Low Priority Queue:
Info:
 Actual Size = 50 Current Size = 0
 Max Size = 0 Full Count = 0
Errors:
 Enqueue Error = 0 Dequeue Error = 0

```

### show telemetry transport

This command displays all configured transport sessions.

```

switch# show telemetry transport

Session Id IP Address Port Encoding Transport Status

0 192.168.20.123 50001 GPB gRPC Connected

```

### show telemetry transport <session-id>

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

```

switch# show telemetry transport 0

Session Id: 0
IP Address:Port 192.168.20.123:50001
Encoding: GPB
Transport: gRPC
Status: Disconnected
Last Connected: Fri Sep 02 11:45:57.505 UTC
Last Disconnected: Never
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
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> stats

This command displays details of a specific transport session.

```

Session Id: 0
Transmission Stats
 Compression: disabled
 Source Interface: not set()
 Transmit Count: 319297
 Last TX time: Fri Aug 02 03:51:15.287 UTC
 Min Tx Time: 1 ms
 Max Tx Time: 3117 ms
 Avg Tx Time: 3 ms
 Cur Tx Time: 1 ms

```

### 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 Errors
Connection Error Count: 0
Transmission Errors
Tx Error Count: 30
Last Tx Error: Thu Aug 01 04:39:47.083 UTC
Last Tx Return Code: No error

```

### show telemetry control databases sensor-paths

These following configuration steps result in the **show telemetry control databases sensor-paths** command output below.

```

feature telemetry

telemetry

```

```

destination-group 1
 ip address 172.25.238.13 port 50600 protocol gRPC encoding GPB
sensor-group 1
 path sys/cdp depth unbounded
 path sys/intf depth unbounded
 path sys/mac depth 0
subscription 1
 dst-grp 1
 snsr-grp 1 sample-interval 1000

```

### Command output.

```
switch# show telemetry control databases sensor-paths
```

```
Sensor Path Database size = 3
```

```

Row ID Subscribed Linked Groups Sec Groups Retrieve level Path(GroupId) :
Query : Filter

```

```
1 No 1 0 Full sys/cdp(1) : NA
: NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 30489/30489/30489
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 6/5/54
Encoding Time in ms (Cur/Min/Max): 5/5/6
Transport Time in ms (Cur/Min/Max): 1027/55/1045
Streaming Time in ms (Cur/Min/Max): 48402/5/48402
```

```
2 No 1 0 Full sys/intf(1) : N
A : NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 539466/539466/539466
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 66/64/114
Encoding Time in ms (Cur/Min/Max): 91/90/92
Transport Time in ms (Cur/Min/Max): 4065/4014/5334
Streaming Time in ms (Cur/Min/Max): 48365/64/48365
```

```
3 No 1 0 Self sys/mac(1) : NA
: NA
```

```
GPB Encoded Data size in bytes (Cur/Min/Max): 247/247/247
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 1/1/47
Encoding Time in ms (Cur/Min/Max): 1/1/1
Transport Time in ms (Cur/Min/Max): 4/1/6
Streaming Time in ms (Cur/Min/Max): 47369/1/47369
```

### show telemetry transport sessions

The following commands loop through all the transport sessions and prints the information in one command:

```

switch# show telemetry transport sessions
switch# show telemetry transport stats
switch# show telemetry transport errors
switch# show telemetry transport all

```

The following is an example for telemetry transport session:

```

switch# show telemetry transport sessions
Session Id: 0
IP Address:Port 172.27.254.13:50004

```

```

Transport: GRPC
Status: Transmit Error
SSL Certificate: trustpoint1
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 2
Last Tx Error: Wed Aug 19 23:32:21.749 UTC
...
Session Id: 4
IP Address:Port 172.27.254.13:50006
Transport: UDP

```

### Telemetry Ephemeral Event

To support ephemeral event, a new sensor path query-condition is added. To enable accounting log ephemeral event streaming, use the following query condition:

```

sensor-group 1
path sys/accounting/log query-condition query-target=subtree&complete-mo=yes¬ify-interval=1

```

The following are the other sensor paths that support ephemeral event:

```

sys/pim/inst/routedb-route, sys/pim/pimifdb-adj, sys/pim/pimifdb-prop
sys/igmp/igmpifdb-prop, sys/igmp/inst/routedb, sys/igmpsnoop/inst/dom/db-extrack,
sys/igmpsnoop/inst/dom/db-group, sys/igmpsnoop/inst/dom/db-mrouter
sys/igmpsnoop/inst/dom/db-querier, sys/igmpsnoop/inst/dom/db-snoop

```

## 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
```

### tmtrace.bin

This BASH shell command collects telemetry traces and prints them out.

```

switch# configure terminal
switch(config)# feature bash
switch(config)# run bash
bash-4.2$ tmtrace.bin -d tm-errors
bash-4.2$ tmtrace.bin -d tm-logs
bash-4.2$ tmtrace.bin -d tm-events

```

For example:

```

bash-4.2$ tmtrace.bin -d tm-logs
[01/25/17 22:52:24.563 UTC 1 29130] [3944724224][tm_ec_dme_auth.c:59] TM_EC: Authentication
refresh url http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.565 UTC 2 29130] [3944724224][tm_ec_dme_rest_util.c:382] TM_EC: Performed
POST request on http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.566 UTC 3 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER: Starting
leaf timer for leaf:0x11e17ea4 time_in_ms:540000

```

```
[01/25/17 22:52:45.317 UTC 4 29130] [3944724224][tm_ec_dme_event_subsc.c:790] TM_EC: Event
subscription database size 0
[01/25/17 22:52:45.317 UTC 5 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER: Starting
leaf timer for leaf:0x11e17e3c time_in_ms:50000
bash-4.2#
```



**Note** The **tm-logs** option is not enabled by default because it is verbose.

Enable **tm-logs** with the `tmtrace.bin -L D tm-logs` command.

Disable **tm-logs** with the `tmtrace.bin -L W tm-logs` command.

### 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 the section "Telemetry Model in the DME." 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 the telemetry feature using the NX-API, you must 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 running Cisco NX-OS Release 7.3(0)I5(1) or a later release.

Your switch must be configured to run the NX-API from the CLI:

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

NX-API sends telemetry data over management VRF:

```
switch(config)# nxapi use-vrf management
```

```
nxapi use-vrf vrf_name
nxapi http port port_number
```

## SUMMARY STEPS

1. Enable the telemetry feature.
2. Create the root level of the JSON payload to describe the telemetry configuration.
3. Create a sensor group to contain the defined sensor paths.
4. (Optional) Add an SSL/TLS certificate and a host.
5. Define a telemetry destination group.
6. Define a telemetry destination profile.
7. Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
8. Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.
9. Create a telemetry subscription to configure the telemetry behavior.
10. Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**).
11. 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.
12. Define one or more sensor paths or nodes to be monitored for telemetry.
13. Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).
14. Add destinations as child objects to the destination group object (**telemetryDestGroup**).
15. Add the destination group object as a child object to the root element (**telemetryEntity**).
16. Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
17. Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
18. Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
19. Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	Enable the telemetry feature. <b>Example:</b> <pre> {   "fmEntity" : {     "children" : [{       "fmTelemetry" : {         "attributes" : {           "adminSt" : "enabled"         }       }     ]   } } </pre>	The root element is <b>fmTelemetry</b> and the base path for this element is <code>sys/fm</code> . Configure the <b>adminSt</b> attribute as <code>enabled</code> .
<b>Step 2</b>	Create the root level of the JSON payload to describe the telemetry configuration. <b>Example:</b> <pre> {   "telemetryEntity": {     "attributes": {       "dn": "sys/tm"     },   } } </pre>	The root element is <b>telemetryEntity</b> and the base path for this element is <code>sys/tm</code> . Configure the <b>dn</b> attribute as <code>sys/tm</code> .
<b>Step 3</b>	Create a sensor group to contain the defined sensor paths. <b>Example:</b> <pre> "telemetrySensorGroup": {   "attributes": {     "id": "10",     "rn": "sensor-10"   },   "dataSrc": "NX-API" }, "children": [{ }] } </pre>	A telemetry sensor group is defined in an object of class <b>telemetrySensorGroup</b> . Configure the following attributes of the object: <ul style="list-style-type: none"> <li>• <b>id</b> — An identifier for the sensor group. Currently only numeric ID values are supported.</li> <li>• <b>rn</b> — The relative name of the sensor group object in the format: <b>sensor-id</b>.</li> <li>• <b>dataSrc</b> — Selects the data source from <b>DEFAULT</b>, <b>DME</b>, <b>YANG</b>, or <b>NX-API</b>.</li> </ul> Children of the sensor group object include sensor paths and one or more relation objects ( <b>telemetryRtSensorGroupRel</b> ) to associate the sensor group with a telemetry subscription.



	Command or Action	Purpose
<b>Step 4</b>	<p>(Optional) Add an SSL/TLS certificate and a host.</p> <p><b>Example:</b></p> <pre>{   "telemetryCertificate": {     "attributes": {       "filename": "root.pem"       "hostname": "c.com"     }   } }</pre>	<p>The <b>telemetryCertificate</b> defines the location of the SSL/TLS certificate with the telemetry subscription/destination.</p>
<b>Step 5</b>	<p>Define a telemetry destination group.</p> <p><b>Example:</b></p> <pre>{   "telemetryDestGroup": {     "attributes": {       "id": "20"     }   } }</pre>	<p>A telemetry destination group is defined in <b>telemetryEntity</b>. Configure the id attribute.</p>
<b>Step 6</b>	<p>Define a telemetry destination profile.</p> <p><b>Example:</b></p> <pre>{   "telemetryDestProfile": {     "attributes": {       "adminSt": "enabled"     },     "children": [       {         "telemetryDestOptSourceInterface": {           "attributes": {             "name": "lo0"           }         }       }     ]   } }</pre>	<p>A telemetry destination profile is defined in <b>telemetryDestProfile</b>.</p> <ul style="list-style-type: none"> <li>• Configure the <b>adminSt</b> attribute as <code>enabled</code>.</li> <li>• Under <b>telemetryDestOptSourceInterface</b>, configure the <b>name</b> attribute with an interface name to stream data from the configured interface to a destination with the source IP address.</li> </ul>
<b>Step 7</b>	<p>Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.</p> <p><b>Example:</b></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 <b>telemetryDest</b>. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> <li>• <b>addr</b> — The IP address of the destination.</li> <li>• <b>port</b> — The port number of the destination.</li> <li>• <b>rn</b> — The relative name of the destination object in the format: <b>path-[path]</b>.</li> <li>• <b>enc</b> — The encoding type of the telemetry data to be sent. NX-OS supports:</li> </ul>

	Command or Action	Purpose
	<pre> } } } </pre>	<ul style="list-style-type: none"> <li>• Google protocol buffers (GPB) for gRPC.</li> <li>• JSON for C.</li> <li>• GPB or JSON for UDP and secure UDP (DTLS).</li> </ul> <p>• <b>proto</b> — The transport protocol type of the telemetry data to be sent. NX-OS supports:</p> <ul style="list-style-type: none"> <li>• gRPC</li> <li>• HTTP</li> <li>• VUDP and secure UDP (DTLS)</li> </ul> <p>• Supported encoded types are:</p> <ul style="list-style-type: none"> <li>• HTTP/JSON YES</li> <li>• HTTP/Form-data YES Only supported for Bin Logging.</li> <li>• GRPC/GPB-Compact YES Native Data Source Only.</li> <li>• GRPC/GPB YES</li> <li>• UDP/GPB YES</li> <li>• UDP/JSON YES</li> </ul>
<b>Step 8</b>	<p>Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.</p> <p><b>Example:</b></p> <pre> {   "telemetryDestGrpOptChunking": {     "attributes": {       "chunkSize": "2048",       "dn": "sys/tm/dest-1/chunking"     }   } } </pre>	<p>See <a href="#">Support for gRPC Chunking, on page 307</a> for more information.</p>
<b>Step 9</b>	<p>Create a telemetry subscription to configure the telemetry behavior.</p> <p><b>Example:</b></p> <pre> "telemetrySubscription": {   "attributes": {     "id": "30",     "rn": "subs-30"   },   "children": [{ </pre>	<p>A telemetry subscription is defined in an object of class <b>telemetrySubscription</b>. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> <li>• <b>id</b> — An identifier for the subscription. Currently only numeric ID values are supported.</li> <li>• <b>rn</b> — The relative name of the subscription object in the format: <b>subs-id</b>.</li> </ul>

	Command or Action	Purpose
	<pre>    }}   }</pre>	Children of the subscription object include relation objects for sensor groups ( <b>telemetryRsSensorGroupRel</b> ) and destination groups ( <b>telemetryRsDestGroupRel</b> ).
<b>Step 10</b>	<p>Add the sensor group object as a child object to the <b>telemetrySubscription</b> element under the root element (<b>telemetryEntity</b>).</p> <p><b>Example:</b></p> <pre>{   "telemetrySubscription": {     "attributes": {       "id": "30"     }     "children": [{       "telemetryRsSensorGroupRel": {         "attributes": {           "sampleIntvl": "5000",           "tDn": "sys/tm/sensor-10"         }       }     ]   } }</pre>	
<b>Step 11</b>	<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><b>Example:</b></p> <pre>"telemetryRsSensorGroupRel": {   "attributes": {     "rType": "mo",     "rn": "rssensorGroupRel-[sys/tm/sensor-10]",     "sampleIntvl": "5000",     "tCl": "telemetrySensorGroup",     "tDn": "sys/tm/sensor-10",     "tType": "mo"   } }</pre>	<p>The relation object is of class <b>telemetryRsSensorGroupRel</b> and is a child object of <b>telemetrySubscription</b>. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> <li>• <b>rn</b> — The relative name of the relation object in the format: <b>rssensorGroupRel-[sys/tm/sensor-group-id]</b>.</li> <li>• <b>sampleIntvl</b> — 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.</li> <li>• <b>tCl</b> — The class of the target (sensor group) object, which is <b>telemetrySensorGroup</b>.</li> <li>• <b>tDn</b> — The distinguished name of the target (sensor group) object, which is <b>sys/tm/sensor-group-id</b>.</li> <li>• <b>rType</b> — The relation type, which is <b>mo</b> for managed object.</li> <li>• <b>tType</b> — The target type, which is <b>mo</b> for managed object.</li> </ul>

	Command or Action	Purpose
<p><b>Step 12</b></p>	<p>Define one or more sensor paths or nodes to be monitored for telemetry.</p> <p><b>Example:</b> Single sensor path</p> <pre data-bbox="251 478 698 856"> {   "telemetrySensorPath": {     "attributes": {       "path": "sys/cdp",       "rn": "path-[sys/cdp]",       "excludeFilter": "",       "filterCondition": "",       "path": "sys/fm/bgp",       "secondaryGroup": "0",       "secondaryPath": "",       "depth": "0",       "alias": "cdp_alias",     }   } } </pre> <p><b>Example:</b> Single sensor path for NX-API</p> <pre data-bbox="251 1018 722 1396"> {   "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><b>Example:</b> Multiple sensor paths</p> <pre data-bbox="251 1558 698 1856"> {   "telemetrySensorPath": {     "attributes": {       "path": "sys/cdp",       "rn": "path-[sys/cdp]",       "excludeFilter": "",       "filterCondition": "",       "path": "sys/fm/bgp",       "secondaryGroup": "0",       "secondaryPath": "",       "depth": "0"     }   } } </pre>	<p>A sensor path is defined in an object of class <b>telemetrySensorPath</b>. Configure the following attributes of the object:</p> <ul style="list-style-type: none"> <li>• <b>path</b> — The path to be monitored.</li> <li>• <b>rn</b> — The relative name of the path object in the format: <b>path-[path]</b></li> <li>• <b>depth</b> — The retrieval level for the sensor path. A depth setting of <b>0</b> retrieves only the root MO properties.</li> <li>• <b>filterCondition</b> — (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: <a href="https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635">https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635</a></li> <li>• <b>alias</b> - Specify an alias for this path.</li> </ul>

	Command or Action	Purpose
	<pre> } }, {   "telemetrySensorPath": {     "attributes": {       "excludeFilter": "",       "filterCondition": "",       "path": "sys/fm/dhcp",       "secondaryGroup": "0",       "secondaryPath": "",       "depth": "0"     }   } } } </pre> <p><b>Example:</b> Single sensor path filtering for BGP disable events:</p> <pre> {   "telemetrySensorPath": {     "attributes": {       "path": "sys/cdp",       "rn": "path-[sys/cdp]",       "excludeFilter": "",       "filterCondition": "eq(fmBgp.operSt.\"disabled\")",       "path": "sys/fm/bgp",       "secondaryGroup": "0",       "secondaryPath": "",       "depth": "0"     }   } } </pre>	
<b>Step 13</b>	Add sensor paths as child objects to the sensor group object ( <b>telemetrySensorGroup</b> ).	
<b>Step 14</b>	Add destinations as child objects to the destination group object ( <b>telemetryDestGroup</b> ).	
<b>Step 15</b>	Add the destination group object as a child object to the root element ( <b>telemetryEntity</b> ).	
<b>Step 16</b>	<p>Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.</p> <p><b>Example:</b></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 <b>telemetryRtSensorGroupRel</b> and is a child object of <b>telemetrySensorGroup</b>. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> <li>• <b>rn</b> — The relative name of the relation object in the format: <b>rtsensorGroupRel-[sys/tm/subscription-id]</b>.</li> <li>• <b>tCl</b> — The target class of the subscription object, which is <b>telemetrySubscription</b>.</li> <li>• <b>tDn</b> — The target distinguished name of the subscription object, which is <b>sys/tm/subscription-id</b>.</li> </ul>

	Command or Action	Purpose
<b>Step 17</b>	<p>Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.</p> <p><b>Example:</b></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 <b>telemetryRtDestGroupRel</b> and is a child object of <b>telemetryDestGroup</b>. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> <li>• <b>rn</b> — The relative name of the relation object in the format: <b>rtdestGroupRel-[sys/tm/subscription-id]</b>.</li> <li>• <b>tCl</b> — The target class of the subscription object, which is <b>telemetrySubscription</b>.</li> <li>• <b>tDn</b> — The target distinguished name of the subscription object, which is <b>sys/tm/subscription-id</b>.</li> </ul>
<b>Step 18</b>	<p>Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.</p> <p><b>Example:</b></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 <b>telemetryRsDestGroupRel</b> and is a child object of <b>telemetrySubscription</b>. Configure the following attributes of the relation object:</p> <ul style="list-style-type: none"> <li>• <b>rn</b> — The relative name of the relation object in the format: <b>rsdestGroupRel-[sys/tm/destination-group-id]</b>.</li> <li>• <b>tCl</b> — The class of the target (destination group) object, which is <b>telemetryDestGroup</b>.</li> <li>• <b>tDn</b> — The distinguished name of the target (destination group) object, which is <b>sys/tm/destination-group-id</b>.</li> <li>• <b>rType</b> — The relation type, which is <b>mo</b> for managed object.</li> <li>• <b>tType</b> — The target type, which is <b>mo</b> for managed object.</li> </ul>
<b>Step 19</b>	<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 that are collected into one POST payload (note that some attributes may not match):

```
{
 "telemetryEntity": {
 "children": [{
 "telemetrySensorGroup": {
 "attributes": {
 "id": "10"
 }
 }
]
 }
}
```

```

 "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

```

For a list of DNs available to the telemetry feature, see *Streaming Telemetry Sources*.

## Telemetry Path Labels

### About Telemetry Path Labels

Beginning with NX-OS release 9.3(1), model-driven telemetry supports path labels. Path labels provide an easy way to gather telemetry data from multiple sources at once. With this feature, you specify the type of telemetry data you want collected, and the telemetry feature gathers that data from multiple paths. The feature then returns the information to one consolidated place, the path label. This feature simplifies using telemetry because you no longer must:

- Have a deep and comprehensive knowledge of the Cisco DME model.
- Create multiple queries and add multiple paths to the subscription, while balancing the number of collected events and the cadence.
- Collect multiple chunks of telemetry information from the switch, which simplifies serviceability.

Path labels span across multiple instances of the same object type in the model, then gather and return counters or events. Path labels support the following telemetry groups:

- Environment, which monitors chassis information, including fan, temperature, power, storage, supervisors, and line cards.
- Interface, which monitors all the interface counters and status changes.

This label supports predefined keyword filters that can refine the returned data by using the **query-condition** command.

- Resources, which monitors system resources such as CPU utilization and memory utilization.

- VXLAN, which monitors VXLAN EVPNs including VXLAN peers, VXLAN counters, VLAN counters, and BGP Peer data.

## Polling for Data or Receiving Events

The sample interval for a sensor group determines how and when telemetry data is transmitted to a path label. The sample interval can be configured either to periodically poll for telemetry data or gather telemetry data when events occur.

- When the sample interval for telemetry is configured as a non-zero value, telemetry periodically sends the data for the environment, interfaces, resources, and VXLAN labels during each sample interval.
- When the sample interval is set to zero, telemetry sends event notifications when the environment, interfaces, resources, and VXLAN labels experience operational state updates, as well as creation and deletion of MOs.

Polling for data or receiving events are mutually exclusive. You can configure polling or event-driven telemetry for each path label.

## Guidelines and Limitations for Path Labels

The telemetry path labels feature has the following guidelines and limitations:

- The feature supports only Cisco DME data source only.
- You cannot mix and match usability paths with regular DME paths in the same sensor group. For example, you cannot configure `sys/intf` and `interface` in the same sensor group. Also, you cannot configure the same sensor group with `sys/intf` and `interface`. If this situation occurs, NX-OS rejects the configuration.
- User filter keywords, such as `oper-speed` and `counters=[detailed]`, are supported only for the `interface` path.
- The feature does not support other sensor path options, such as `depth` or `filter-condition`.

## Configuring the Interface Path to Poll for Data or Events

The interface path label monitors all the interface counters and status changes. It supports the following interface types:

- Physical
- Subinterface
- Management
- Loopback
- VLAN
- Port Channel

You can configure the interface path label to either periodically poll for data or receive events. See [Polling for Data or Receiving Events, on page 347](#).



**Note** The model does not support counters for subinterface, loopback, or VLAN, so they are not streamed out.

## SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **path interface**
5. **destination-group** *grp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

## DETAILED STEPS

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(config-telemetry)# sensor-group 6 switch-1(conf-tm-sensor)#</pre>	Create a sensor group for telemetry data.
<b>Step 4</b>	<b>path interface</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path interface switch-1(conf-tm-sensor)#</pre>	<p>Configure the interface path label, which enables sending one telemetry data query for multiple individual interfaces. The label consolidates the queries for multiple interfaces into one. Telemetry then telemetry gathers the data and returns it to the label.</p> <p>Depending on how the polling interval is configured, interface data is sent based on a periodic basis or whenever the interface state changes.</p>

	Command or Action	Purpose
Step 5	<b>destination-group</b> <i>grp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sensor) # destination-group 33 switch-1(conf-tm-dest) #</pre>	Enter telemetry destination group submode and configure the destination group.
Step 6	<b>ip address</b> <i>ip_addr</i> <b>port</b> <i>port</i> <b>Example:</b> <pre>switch-1(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest) #</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
Step 7	<b>subscription</b> <i>sub_id</i> <b>Example:</b> <pre>switch-1(conf-tm-dest) # subscription 33 switch-1(conf-tm-sub) #</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
Step 8	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub) #</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
Step 9	<b>dst-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # dst-grp 33 switch-1(conf-tm-sub) #</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Interface Path for Non-Zero Counters

You can configure the interface path label with a predefined keyword filter that returns only counters that have nonzero values. The filter is `counters=[detailed]`.

By using this filter, the interface path gathers all the available interface counters, filters the collected data, then forwards the results to the receiver. The filter is optional, and if you do not use it, all counters, including zero-value counters, are displayed for the interface path.



**Note** Using the filter is conceptually similar to issuing **show interface mgmt0 counters detailed**

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **path interface query-condition** `counters=[detailed]`
5. **destination-group** *grp\_id*

6. `ip address ip_addr port port`
7. `subscription sub_id`
8. `snsr-group sgrp_id sample-interval interval`
9. `dst-group dgrp_id`

## DETAILED STEPS

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group sgrp_id</b> <b>Example:</b> <pre>switch-1(config-telemetry)# sensor-group 6 switch-1(conf-tm-sensor)#</pre>	Create a sensor group for telemetry data.
<b>Step 4</b>	<b>path interface query-condition counters=[detailed]</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed] switch-1(conf-tm-sensor)#</pre>	Configure the interface path label and query for only the nonzero counters from all interfaces.
<b>Step 5</b>	<b>destination-group grp_id</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 6</b>	<b>ip address ip_addr port port</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
<b>Step 7</b>	<b>subscription sub_id</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.



	Command or Action	Purpose
Step 8	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub) #</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
Step 9	<b>dst-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # dst-grp 33 switch-1(conf-tm-sub) #</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Interface Path for Operational Speeds

You can configure the interface path label with a pre-defined keyword filter that returns counters for interfaces of specified operational speeds. The filter is `oper-speed=[ ]`. The following operational speeds are supported: auto, 10M, 100M, 1G, 10G, 40G, 200G, and 400G.

By using this filter, the interface path gathers the telemetry data for interfaces of the specified speed, then forwards the results to the receiver. The filter is optional. If you do not use it, counters for all interfaces are displayed, regardless of their operational speed.

The filter can accept multiple speeds as a comma-separated list, for example `oper-speed=[1G,10G]` to retrieve counters for interfaces that operate at 1 and 10 Gbps. Do not use a blank space as a delimiter.



**Note** Interface types subinterface, loopback, and VLAN do not have operational speed properties, so the filter does not support these interface types.

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **snsr-group** *sgrp\_id* **sample-interval** *interval*
4. **path interface query-condition oper-speed=[speed]**
5. **destination-group** *dgrp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>snsr-group <i>sgrp_id</i> sample-interval <i>interval</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
<b>Step 4</b>	<b>path interface query-condition oper-speed=[<i>speed</i>]</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G] switch-1(conf-tm-sensor)#</pre>	Configure the interface path label and query for counters from interfaces running the specified speed, which in this example, is 1 and 40 Gbps only.
<b>Step 5</b>	<b>destination-group <i>grp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 6</b>	<b>ip address <i>ip_addr</i> port <i>port</i></b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
<b>Step 7</b>	<b>subscription <i>sub_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 8</b>	<b>snsr-group <i>sgrp_id</i> sample-interval <i>interval</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.

	Command or Action	Purpose
Step 9	<b>dst-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# <b>dst-grp</b> 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Interface Path with Multiple Queries

You can configure multiple filters for the same query condition in the interface path label. When you do so, the individual filters you use are ANDed.

Separate each filter in the query condition by using a comma. You can specify any number of filters for the query-condition, but the more filters you add, the more focused the results become.

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **path interface query-condition** counters=[detailed],oper-speed=[1G,40G]
5. **destination-group** *grp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
Step 1	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# <b>configure terminal</b> switch-1(config)#</pre>	Enter configuration mode.
Step 2	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# <b>telemetry</b> switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
Step 3	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(config-telemetry)# <b>sensor-group</b> 6 switch-1(conf-tm-sensor)#</pre>	Create a sensor group for telemetry data.

	Command or Action	Purpose
<b>Step 4</b>	<b>path interface query-condition</b> <b>counters=[detailed],oper-speed=[1G,40G]</b>  <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G] switch-1(conf-tm-sensor)#</pre>	Configures multiple conditions in the same query. In this example, the query does both of the following: <ul style="list-style-type: none"> <li>• Gathers and returns non-zero counters on interfaces running at 1 Gbps.</li> <li>• Gathers and returns non-zero counters on interfaces running at 40 Gbps.</li> </ul>
<b>Step 5</b>	<b>destination-group grp_id</b>  <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 6</b>	<b>ip address ip_addr port port</b>  <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
<b>Step 7</b>	<b>subscription sub_id</b>  <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 8</b>	<b>snsr-group sgrp_id sample-interval interval</b>  <b>Example:</b> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
<b>Step 9</b>	<b>dst-group dgrp_id</b>  <b>Example:</b> <pre>switch-1(conf-tm-sub)# dst-grp 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Environment Path to Poll for Data or Events

The environment path label monitors chassis information, including fan, temperature, power, storage, supervisors, and line cards. You can configure the environment path to either periodically poll for telemetry data or get the data when events occur. For information, see [Polling for Data or Receiving Events, on page 347](#).

You can set the resources path to return system resource information through either periodic polling or based on events. This path does not support filtering.

## SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **path environment**
5. **destination-group** *grp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
Step 1	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
Step 2	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
Step 3	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(config-telemetry)# sensor-group 6 switch-1(conf-tm-sensor)#</pre>	Create a sensor group for telemetry data.
Step 4	<b>path environment</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path environment switch-1(conf-tm-sensor)#</pre>	<p>Configures the environment path label, which enables telemetry data for multiple individual environment objects to be sent to the label. The label consolidates the multiple data inputs into one output.</p> <p>Depending on the sample interval, the environment data is either streaming based on the polling interval, or sent when events occur.</p>
Step 5	<b>destination-group</b> <i>grp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
Step 6	<b>ip address</b> <i>ip_addr</i> <b>port</b> <i>port</i> <b>Example:</b>	Configure the telemetry data for the subscription to stream to the specified IP address and port.

	Command or Action	Purpose
	<pre>switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest)#</pre>	
<b>Step 7</b>	<p><b>subscription</b> <i>sub_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 8</b>	<p><b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when environment events occur.
<b>Step 9</b>	<p><b>dst-group</b> <i>dgrp_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# dst-grp 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Resources Path for Poll for Events or Data

The resources path monitors system resources such as CPU utilization and memory utilization. You can configure this path to either periodically gather telemetry data, or when events occur. See [Polling for Data or Receiving Events, on page 347](#).

This path does not support filtering.

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **path resources**
5. **destination-group** *grp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
Step 1	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
Step 2	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
Step 3	<b>sensor-group <i>sgrp_id</i></b> <b>Example:</b> <pre>switch-1(config-telemetry)# sensor-group 6 switch-1(conf-tm-sensor)#</pre>	Create a sensor group for telemetry data.
Step 4	<b>path resources</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# path resources switch-1(conf-tm-sensor)#</pre>	<p>Configure the resources path label, which enables telemetry data for multiple individual system resources to be sent to the label. The label consolidates the multiple data inputs into one output.</p> <p>Depending on the sample interval, the resource data is either streaming based on the polling interval, or sent when system memory changes to Not OK.</p>
Step 5	<b>destination-group <i>grp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
Step 6	<b>ip address <i>ip_addr</i> port <i>port</i></b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
Step 7	<b>subscription <i>sub_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
Step 8	<b>snsr-group <i>sgrp_id</i> sample-interval <i>interval</i></b> <b>Example:</b>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when resource events occur.

	Command or Action	Purpose
	<pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	
<b>Step 9</b>	<p><b>dst-group</b> <i>dgrp_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# dst-grp 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the VXLAN Path to Poll for Events or Data

The VXLAN path label provides information about the switch's Virtual Extensible LAN EVPNs, including VXLAN peers, VXLAN counters, VLAN counters, and BGP Peer data. You can configure this path label to gather telemetry information either periodically, or when events occur. See [Polling for Data or Receiving Events, on page 347](#).

This path does not support filtering.

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **vxlan environment**
5. **destination-group** *grp\_id*
6. **ip address** *ip\_addr* **port** *port*
7. **subscription** *sub\_id*
8. **snsr-group** *sgrp\_id* **sample-interval** *interval*
9. **dst-group** *dgrp\_id*

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<p><b>configure terminal</b></p> <p><b>Example:</b></p> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<p><b>telemetry</b></p> <p><b>Example:</b></p> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.



	Command or Action	Purpose
Step 3	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(config-telemetry) # sensor-group 6 switch-1(conf-tm-sensor) #</pre>	Create a sensor group for telemetry data.
Step 4	<b>vxlan environment</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor) # vxlan environment switch-1(conf-tm-sensor) #</pre>	Configure the VXLAN path label, which enables telemetry data for multiple individual VXLAN objects to be sent to the label. The label consolidates the multiple data inputs into one output. Depending on the sample interval, the VXLAN data is either streaming based on the polling interval, or sent when events occur.
Step 5	<b>destination-group</b> <i>grp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sensor) # destination-group 33 switch-1(conf-tm-dest) #</pre>	Enter telemetry destination group submode and configure the destination group.
Step 6	<b>ip address</b> <i>ip_addr</i> <b>port</b> <i>port</i> <b>Example:</b> <pre>switch-1(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch-1(conf-tm-dest) #</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port.
Step 7	<b>subscription</b> <i>sub_id</i> <b>Example:</b> <pre>switch-1(conf-tm-dest) # subscription 33 switch-1(conf-tm-sub) #</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
Step 8	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub) #</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when VXLAN events occur.
Step 9	<b>dst-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub) # dst-grp 33 switch-1(conf-tm-sub) #</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Verifying the Path Label Configuration

At any time, you can verify that path labels are configured, and check their values by displaying the running telemetry configuration.

### SUMMARY STEPS

1. **show running-config-telemetry**

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
Step 1	<p><b>show running-config-telemetry</b></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sensor)# show running-config telemetry  !Command: show running-config telemetry !Running configuration last done at: Mon Jun 10 08:10:17 2019 !Time: Mon Jun 10 08:10:17 2019  version 9.3(1) Bios:version feature telemetry  telemetry   destination-profile   use-nodeid tester   sensor-group 4     path interface query-condition and(counters=[detailed],oper-speed=[1G,10G])   sensor-group 6     path interface query-condition oper-speed=[1G,40G]   subscription 6     snsr-grp 6 sample-interval 6000 nxosv2(conf-tm-sensor)#</pre>	<p>Displays the current running config for telemetry,</p> <p>In this example, sensor group 4 is configured to gather non-zero counters from interfaces running at 1 and 10 Gbps. Sensor group 6 is configured to gather all counters from interfaces running at 1 and 40 Gbps.</p>

## Displaying Path Label Information

## Path Label Show Commands

Through the **show telemetry usability** commands, you can display the individual paths that the path label walks when you issue a query.

Command	Shows
<b>show telemetry usability {all   environment   interface   resources   vxlan}</b>	<p>Either all telemetry paths for all path labels, or all telemetry paths for a specified path label. Also, the output shows whether each path reports telemetry data based on periodic polling or events.</p> <p>For the interfaces path label, also any keyword filters or query conditions you configured.</p>
<b>show running-config telemetry</b>	The running configuration for telemetry and selected path information.





# Native Data Source Paths

## About Native Data Source Paths

NX-OS Telemetry supports the native data source, which is a neutral data source that is not restricted to a specific infrastructure or database. Instead, the native data source enables components or applications to hook into and inject relevant information into the outgoing telemetry stream. This feature provides flexibility because the path for the native data source does not belong to any infrastructure, so any native applications can interact with NX-OS Telemetry.

The native data source path enables you to subscribe to specific sensor paths to receive selected telemetry data. The feature works with the NX-SDK to support streaming telemetry data from the following paths:

- RIB path, which sends telemetry data for the IP routes.
- MAC path, which sends telemetry data for static and dynamic MAC entries.
- Adjacency path, which sends telemetry data for IPv4 and IPv6 adjacencies.

When you create a subscription, all telemetry data for the selected path streams to the receiver as a baseline. After the baseline, only event notifications stream to the receiver.

Streaming of native data source paths supports the following encoding types:

- Google Protobuf (GPB)
- JavaScript Object Notation (JSON)
- Compact Google Protobuf (compact GPB)

## Telemetry Data Streamed for Native Data Source Paths

For each source path, the following table shows the information that is streamed when the subscription is first created (the baseline) and when event notifications occur.

Path Type	Subscription Baseline	Event Notifications
RIB	Sends all routes	<p>Sends event notifications for create, update, and delete events. The following values are exported through telemetry for the RIB path:</p> <ul style="list-style-type: none"> <li>• Next-hop routing information: <ul style="list-style-type: none"> <li>• Address of the next hop</li> <li>• Outgoing interface for the next hop</li> <li>• VRF name for the next hop</li> <li>• Owner of the next hop</li> <li>• Preference for the next hop</li> <li>• Metric for the next hop</li> <li>• Tag for the next hop</li> <li>• Segment ID for the next hop</li> <li>• Tunnel ID for the next hop</li> <li>• Encapsulation type for the next hop</li> <li>• Bitwise OR of flags for the Next Hop Type</li> </ul> </li> <li>• For Layer-3 routing information: <ul style="list-style-type: none"> <li>• VRF name of the route</li> <li>• Route prefix address</li> <li>• Mask length for the route</li> <li>• Number of next hops for the route</li> <li>• Event type</li> <li>• Next hops</li> </ul> </li> </ul>

Path Type	Subscription Baseline	Event Notifications
MAC	Executes a <code>GETALL</code> from DME for static and dynamic MAC entries	<p>Sends event notifications for add, update, and delete events. The following values are exported through telemetry for the MAC path:</p> <ul style="list-style-type: none"> <li>• MAC address</li> <li>• MAC address type</li> <li>• VLAN number</li> <li>• Interface name</li> <li>• Event types</li> </ul> <p>Both static and dynamic entries are supported in event notifications.</p>
Adjacency	Sends the IPv4 and IPv6 adjacencies	<p>Sends event notifications for add, update, and delete events. The following values are exported through telemetry for the Adjacency path:</p> <ul style="list-style-type: none"> <li>• IP address</li> <li>• MAC address</li> <li>• Interface name</li> <li>• Physical interface name</li> <li>• VRF name</li> <li>• Preference</li> <li>• Source for the adjacency</li> <li>• Address family for the adjacency</li> <li>• Adjacency event type</li> </ul>

For additional information, refer to Github <https://github.com/CiscoDevNet/nx-telemetry-proto>.

## Guidelines and Limitations for Native Data Source Path

The native data source path feature has the following guidelines and limitations:

- For streaming from the RIB, MAC, and Adjacency native data source paths, sensor-path property updates do not support custom criteria like **depth**, **query-condition**, or **filter-condition**.
- Beginning with Cisco NX-OS Release 10.4(3)F, new query conditions are introduced to support sample-based subscription or updates-only support for RIB native path.

## Configuring the Native Data Source Path for Routing Information

You can configure the native data source path for routing information, which sends information about all routes that are contained in the URIB. When you subscribe, the baseline sends all the route information. After the baseline, notifications are sent for route update and delete operations for the routing protocols that the switch supports. For the data sent in the RIB notifications, see [Telemetry Data Streamed for Native Data Source Paths](#), on page 363.

### Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **data-source native**
5. **path rib query-condition** [**data=ephemeral** | **updates\_only**]
6. **destination-group** *grp\_id*
7. **ip address** *ip\_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub\_id*
9. **snsr-group** *sgrp\_id* **sample-interval** *interval*
10. **dst-group** *dgrp\_id*

### DETAILED STEPS

#### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# sensor-grp 6 switch-1(conf-tm-sub)#</pre>	Create a sensor group.



	Command or Action	Purpose
Step 4	<b>data-source native</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# data-source native switch-1(conf-tm-sensor)#</pre>	Set the data source to native so that any native application can use the streamed data without requiring a specific model or database.
Step 5	<b>path rib query-condition [data=ephemeral   updates_only]</b> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path rib nxosv2(conf-tm-sensor)#</pre> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path rib query condition data=ephemeral nxosv2(conf-tm-sensor)#</pre> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path rib query condition updates_only nxosv2(conf-tm-sensor)#</pre>	Configure the RIB path which streams routes and route update information.  <b>query condition data=ephemeral</b> (optional) - You can configure sample interval 0 or other than 0. This sample interval will determine how frequently the route information is sent to the destination periodically (at the configured sample interval).  <b>query condition updates-only</b> (optional) - Supported only for sample interval 0. With this query condition, the initial snapshot data will not be sent, only the route information updates will be sent to the destination.
Step 6	<b>destination-group grp_id</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
Step 7	<b>ip address ip_addr port port protocol { HTTP   gRPC } encoding { JSON   GPB   GPB-compact }</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch-1(conf-tm-dest)#</pre> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch-1(conf-tm-dest)#</pre> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port and set the protocol and encoding for the data stream.
Step 8	<b>subscription sub_id</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
Step 9	<b>snsr-group sgrp_id sample-interval interval</b> <b>Example:</b>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling

	Command or Action	Purpose
	<pre>switch-1 (conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch-1 (conf-tm-sub) #</pre>	<p>interval determines whether the switch sends telemetry data periodically, or when rib events occur.</p> <p><b>Note</b> Depending on the sample interval, the rib sensor path streams based on the polling interval.</p>
<b>Step 10</b>	<p><b>dst-group</b> <i>dgrp_id</i></p> <p><b>Example:</b></p> <pre>switch-1 (conf-tm-sub) # dst-grp 33 switch-1 (conf-tm-sub) #</pre>	<p>Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.</p>

## Configuring the Native Data Source Path for MAC Information

You can configure the native data source path for MAC information, which sends information about all entries in the MAC table. When you subscribe, the baseline sends all the MAC information. After the baseline, notifications are sent for add, update, and delete MAC address operations. For the data sent in the MAC notifications, see [Telemetry Data Streamed for Native Data Source Paths, on page 363](#).



**Note** For update or delete events, MAC notifications are sent only for the MAC addresses that have IP adjacencies.

### Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **data-source native**
5. **path mac**
6. **destination-group** *grp\_id*
7. **ip address** *ip\_addr* **port** *port* **protocol** { HTTP | gRPC } **encoding** { JSON | GPB | GPB-compact }
8. **subscription** *sub\_id*
9. **snsr-group** *sgrp\_id* **sample-interval** *interval*
10. **dst-group** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group <i>sgrp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sub)# sensor-grp 6 switch-1(conf-tm-sub)#</pre>	Create a sensor group.
<b>Step 4</b>	<b>data-source native</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# data-source native switch-1(conf-tm-sensor)#</pre>	Set the data source to native so that any native application can use the streamed data without requiring a specific model or database.
<b>Step 5</b>	<b>path mac</b> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path mac nxosv2(conf-tm-sensor)#</pre>	Configure the MAC path which streams information about MAC entries and MAC notifications.
<b>Step 6</b>	<b>destination-group <i>grp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 7</b>	<b>ip address <i>ip_addr</i> port <i>port</i> protocol { HTTP   gRPC } encoding { JSON   GPB   GPB-compact }</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch-1(conf-tm-dest)#</pre> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch-1(conf-tm-dest)#</pre> <b>Example:</b>	Configure the telemetry data for the subscription to stream to the specified IP address and port and set the protocol and encoding for the data stream.

	Command or Action	Purpose
	switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch-1(conf-tm-dest)#	
<b>Step 8</b>	<b>subscription</b> <i>sub_id</i>  <b>Example:</b> switch-1(conf-tm-dest)# <b>subscription</b> 33 switch-1(conf-tm-sub)#	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 9</b>	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i>  <b>Example:</b> switch-1(conf-tm-sub)# <b>snsr-grp</b> 6 <b>sample-interval</b> 5000 switch-1(conf-tm-sub)#	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
<b>Step 10</b>	<b>dst-group</b> <i>dgrp_id</i>  <b>Example:</b> switch-1(conf-tm-sub)# <b>dst-grp</b> 33 switch-1(conf-tm-sub)#	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Native Data Source Path for All MAC Information

You can configure the native data source path for MAC information, which sends information about all entries in the MAC table from Layer 3 and Layer 2. When you subscribe, the baseline sends all the MAC information. After the baseline, notifications are sent for add, update, and delete MAC address operations. For the data sent in the MAC notifications, see [Telemetry Data Streamed for Native Data Source Paths, on page 363](#).



**Note** For update or delete events, MAC notifications are sent only for the MAC addresses that have IP adjacencies.

### Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **data-source native**
5. **path mac-all**
6. **destination-group** *dgrp\_id*
7. **ip address** *ip\_addr* **port** *port* **protocol** { HTTP | gRPC } **encoding** { JSON | GPB | GPB-compact }
8. **subscription** *sub\_id*
9. **snsr-group** *sgrp\_id* **sample-interval** *interval*

10. `dst-group` *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
Step 1	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
Step 2	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
Step 3	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# sensor-grp 6 switch-1(conf-tm-sub)#</pre>	Create a sensor group.
Step 4	<b>data-source native</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# data-source native switch-1(conf-tm-sensor)#</pre>	Set the data source to native so that any native application can use the streamed data without requiring a specific model or database.
Step 5	<b>path mac-all</b> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path mac-all nxosv2(conf-tm-sensor)#</pre>	Configure the MAC path which streams information about all MAC entries and MAC notifications.
Step 6	<b>destination-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
Step 7	<b>ip address</b> <i>ip_addr</i> <b>port</b> <i>port</i> <b>protocol</b> { HTTP   gRPC } <b>encoding</b> { JSON   GPB   GPB-compact } <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch-1(conf-tm-dest)#</pre> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port and set the protocol and encoding for the data stream.

	Command or Action	Purpose
	<b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch-1(conf-tm-dest)#</pre>	
<b>Step 8</b>	<b>subscription</b> <i>sub_id</i> <b>Example:</b> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 9</b>	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
<b>Step 10</b>	<b>dst-group</b> <i>dgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# dst-grp 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Configuring the Native Data Path for IP Adjacencies

You can configure the native data source path for IP adjacency information, which sends information about all IPv4 and IPv6 adjacencies for the switch. When you subscribe, the baseline sends all the adjacencies. After the baseline, notifications are sent for add, update, and delete adjacency operations. For the data sent in the adjacency notifications, see [Telemetry Data Streamed for Native Data Source Paths, on page 363](#).

### Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

### SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **data-source native**
5. **path adjacency**
6. **destination-group** *dgrp\_id*
7. **ip address** *ip\_addr* **port** *port* **protocol** { HTTP | gRPC } **encoding** { JSON | GPB | GPB-compact }
8. **subscription** *sub\_id*
9. **snsr-group** *sgrp\_id* **sample-interval** *interval*
10. **dst-group** *dgrp\_id*

## DETAILED STEPS

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group <i>sgrp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sub)# sensor-grp 6 switch-1(conf-tm-sub)#</pre>	Create a sensor group.
<b>Step 4</b>	<b>data-source native</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# data-source native switch-1(conf-tm-sensor)#</pre>	Set the data source to native so that any native application can use the streamed data.
<b>Step 5</b>	<b>path adjacency</b> <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path adjacency nxosv2(conf-tm-sensor)#</pre>	Configure the Adjacency path which streams information about the IPv4 and IPv6 adjacencies.
<b>Step 6</b>	<b>destination-group <i>grp_id</i></b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 7</b>	<b>ip address <i>ip_addr</i> port <i>port</i> protocol { HTTP   gRPC } encoding { JSON   GPB   GPB-compact }</b> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch-1(conf-tm-dest)#</pre> <b>Example:</b> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch-1(conf-tm-dest)#</pre> <b>Example:</b>	Configure the telemetry data for the subscription to stream to the specified IP address and port and set the protocol and encoding for the data stream.

	Command or Action	Purpose
	switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch-1(conf-tm-dest)#	
<b>Step 8</b>	<b>subscription</b> <i>sub_id</i>  <b>Example:</b> switch-1(conf-tm-dest)# <b>subscription</b> 33 switch-1(conf-tm-sub)#	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 9</b>	<b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i>  <b>Example:</b> switch-1(conf-tm-sub)# <b>snsr-grp</b> 6 <b>sample-interval</b> 5000 switch-1(conf-tm-sub)#	Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when interface events occur.
<b>Step 10</b>	<b>dst-group</b> <i>dgrp_id</i>  <b>Example:</b> switch-1(conf-tm-sub)# <b>dst-grp</b> 33 switch-1(conf-tm-sub)#	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Displaying Native Data Source Path Information

Use the NX-OS **show telemetry event collector** commands to display statistics and counters, or errors for the native data source path.

### Displaying Statistics

You can issue **show telemetry event collector stats** command to display the statistics and counters for each native data source path.

An example of statistics for the RIB path:

```
switch-1# show telemetry event collector stats
```

```

Row ID Collection Count Latest Collection Time Sensor Path(GroupId)

1 4 Mon Jul 01 13:53:42.384 PST rib(1)
switch-1#
```

An example of the statistics for the MAC path:

```
switch-1# show telemetry event collector stats
```

```

Row ID Collection Count Latest Collection Time Sensor Path(GroupId)

1 3 Mon Jul 01 14:01:32.161 PST mac(1)
switch-1#
```

An example of the statistics for the Adjacency path:

```
switch-1# show telemetry event collector stats
```

```

```



```

Row ID Collection Count Latest Collection Time Sensor Path(GroupId)

1 7 Mon Jul 01 14:47:32.260 PST adjacency(1)
switch-1#

```

### Displaying Error Counters

You can use the **show telemetry event collector stats** command to display the error totals for all the native data source paths.

```
switch-1# show telemetry event collector errors
```

```

-
Error Description Error Count

-
Dme Event Subscription Init Failures - 0
Event Data Enqueue Failures - 0
Event Subscription 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
switch-1#

```

## Streaming Syslog

### About Streaming Syslog for Telemetry

Beginning with Cisco NX-OS release 9.3(3), model-driven telemetry supports streaming of syslogs using YANG as a data source. When you create a subscription, all the syslogs are streamed to the receiver as a baseline. This feature works with the NX-SDK to support streaming syslog data from the following syslog paths:

- Cisco-NX-OS-Syslog-oper:syslog
- Cisco-NX-OS-Syslog-oper:syslog/messages

After the baseline, only syslog event notifications stream to the receiver. Streaming of syslog paths supports the following encoding types:

- Google Protobuf (GPB)
- JavaScript Object Notation (JSON)

### Configuring the Native Data Source Path for Routing Information

You can configure the native data source path for routing information, which sends information about all routes that are contained in the URIB. When you subscribe, the baseline sends all the route information. After the baseline, notifications are sent for route update and delete operations for the routing protocols that the switch supports. For the data sent in the RIB notifications, see [Telemetry Data Streamed for Native Data Source Paths, on page 363](#).

**Before you begin**

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

**SUMMARY STEPS**

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp\_id*
4. **data-source native**
5. **path rib query-condition** [**data=ephemeral** | **updates\_only**]
6. **destination-group** *grp\_id*
7. **ip address** *ip\_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub\_id*
9. **snsr-group** *sgrp\_id* **sample-interval** *interval*
10. **dst-group** *dgrp\_id*

**DETAILED STEPS****Procedure**

	<b>Command or Action</b>	<b>Purpose</b>
<b>Step 1</b>	<b>configure terminal</b> <b>Example:</b> <pre>switch-1# configure terminal switch-1(config)#</pre>	Enter configuration mode.
<b>Step 2</b>	<b>telemetry</b> <b>Example:</b> <pre>switch-1(config)# telemetry switch-1(config-telemetry)#</pre>	Enter configuration mode for the telemetry features.
<b>Step 3</b>	<b>sensor-group</b> <i>sgrp_id</i> <b>Example:</b> <pre>switch-1(conf-tm-sub)# sensor-grp 6 switch-1(conf-tm-sub)#</pre>	Create a sensor group.
<b>Step 4</b>	<b>data-source native</b> <b>Example:</b> <pre>switch-1(conf-tm-sensor)# data-source native switch-1(conf-tm-sensor)#</pre>	Set the data source to native so that any native application can use the streamed data without requiring a specific model or database.
<b>Step 5</b>	<b>path rib query-condition</b> [ <b>data=ephemeral</b>   <b>updates_only</b> ] <b>Example:</b> <pre>nxosv2(conf-tm-sensor)# path rib nxosv2(conf-tm-sensor)#</pre>	Configure the RIB path which streams routes and route update information.  <b>query condition data=ephemeral</b> (optional) - You can configure sample interval 0 or other than 0. This sample interval will determine how frequently the route

	Command or Action	Purpose
	<p><b>Example:</b></p> <pre>nxosv2(conf-tm-sensor)# path rib query condition data=ephemeral nxosv2(conf-tm-sensor)#</pre> <p><b>Example:</b></p> <pre>nxosv2(conf-tm-sensor)# path rib query condition updates_only nxosv2(conf-tm-sensor)#</pre>	<p>information is sent to the destination periodically (at the configured sample interval).</p> <p><b>query condition updates-only</b> (optional) - Supported only for sample interval 0. With this query condition, the initial snapshot data will not be sent, only the route information updates will be sent to the destination.</p>
<b>Step 6</b>	<p><b>destination-group</b> <i>grp_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sensor)# destination-group 33 switch-1(conf-tm-dest)#</pre>	Enter telemetry destination group submode and configure the destination group.
<b>Step 7</b>	<p><b>ip address</b> <i>ip_addr</i> <b>port</b> <i>port</i> <b>protocol</b> { HTTP   gRPC } <b>encoding</b> { JSON   GPB   GPB-compact }</p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch-1(conf-tm-dest)#</pre> <p><b>Example:</b></p> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch-1(conf-tm-dest)#</pre> <p><b>Example:</b></p> <pre>switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch-1(conf-tm-dest)#</pre>	Configure the telemetry data for the subscription to stream to the specified IP address and port and set the protocol and encoding for the data stream.
<b>Step 8</b>	<p><b>subscription</b> <i>sub_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-dest)# subscription 33 switch-1(conf-tm-sub)#</pre>	Enter telemetry subscription submode, and configure the telemetry subscription.
<b>Step 9</b>	<p><b>snsr-group</b> <i>sgrp_id</i> <b>sample-interval</b> <i>interval</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch-1(conf-tm-sub)#</pre>	<p>Link the sensor group to the current subscription and set the data sampling interval in milliseconds. The sampling interval determines whether the switch sends telemetry data periodically, or when rib events occur.</p> <p><b>Note</b> Depending on the sample interval, the rib sensor path streams based on the polling interval.</p>
<b>Step 10</b>	<p><b>dst-group</b> <i>dgrp_id</i></p> <p><b>Example:</b></p> <pre>switch-1(conf-tm-sub)# dst-grp 33 switch-1(conf-tm-sub)#</pre>	Link the destination group to the current subscription. The destination group that you specify must match the destination group that you configured in the <b>destination-group</b> command.

## Telemetry Data Streamed for Syslog Path

For each source path, the following table shows the information that is streamed when the subscription is first created "the baseline" and when event notifications occur.

Path	Subscription Baseline	Event Notification
Cisco-NX-OS-Syslog-oper:syslog/messages	Stream all the existing syslogs from the switch.	Sends event notification for syslog occurred on the switch: <ul style="list-style-type: none"> <li>• message-id</li> <li>• node-name</li> <li>• time-stamp</li> <li>• time-of-day</li> <li>• time-zone</li> <li>• category</li> <li>• message-name</li> <li>• severity</li> <li>• text</li> </ul>

### Displaying Syslog Path Information

Use the Cisco NX-OS **show telemetry event collector** commands to display statistics and counters, or errors for the syslog path.

### Displaying Statistics

You can enter the **show telemetry event collector stats** command to display the statistics and counters for each syslog path.

The following is an example of statistics for the syslog path:

```
switch# show telemetry event collector stats

Row ID Collection Count Latest Collection Time Sensor Path(GroupId)

1 138 Tue Dec 03 11:20:08.200 PST Cisco-NX-OS-Syslog-oper:syslog(1)

2 138 Tue Dec 03 11:20:08.200 PST Cisco-NX-OS-Syslog-oper:syslog/messages(1)
```

### Displaying Error Counters

You can use the **show telemetry event collector errors** command to display the error totals for all the syslog paths.

```
switch(config-if)# show telemetry event collector errors

Error Description Error Count
```

```

Dme Event Subscription Init Failures - 0
Event Data Enqueue Failures - 0
Event Subscription 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

```

## Sample JSON Output

The following is a sample of JSON output:

```

172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
>>> URL : /network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages
>>> TM-HTTP-VER : 1.0.0
>>> TM-HTTP-CNT : 1
>>> Content-Type : application/json
>>> Content-Length : 578
 Path => Cisco-NX-OS-Syslog-oper:syslog/messages
 node_id_str : task-n9k-1
 collection_id : 40
 data_source : YANG
 data :

[
 [
 {
 "message-id": 420
 },
 {
 "category": "ETHPORT",
 "group": "ETHPORT",
 "message-name": "IF_UP",
 "node-name": "task-n9k-1",
 "severity": 5,
 "text": "Interface loopback10 is up ",
 "time-of-day": "Dec 3 2019 11:38:51",
 "time-stamp": "1575401931000",
 "time-zone": ""
 }
]
]

```

.

## Sample KVGPB Output

The following is a sample KVGPB output.

```

KVGPB Output:
---Telemetry msg received @ 18:22:04 UTC

```

```
Read frag:1 size:339 continue to block on read..
All the fragments:1 read successfully total size read:339
node_id_str: "task-n9k-1"
subscription_id_str: "1"
collection_id: 374
data_gpbkv {
 fields {
 name: "keys"
 fields {
 name: "message-id"
 uint32_value: 374
 }
 }
 fields {
 name: "content"
 fields {
 fields {
 name: "node-name"
 string_value: "task-n9k-1"
 }
 fields {
 name: "time-of-day"
 string_value: "Jun 26 2019 18:20:21"
 }
 fields {
 name: "time-stamp"
 uint64_value: 1574293838000
 }
 fields {
 name: "time-zone"
 string_value: "UTC"
 }
 }
 }
}
```

```
fields {
 name: "process-name"
 string_value: ""
}
fields {
 name: "category"
 string_value: "VSHD"
}
fields {
 name: "group"
 string_value: "VSHD"
}
fields {
 name: "message-name"
 string_value: "VSHD_SYSLOG_CONFIG_I"
}
fields {
 name: "severity"
 uint32_value: 5
}
fields {
 name: "text"
 string_value: "Configured from vty by admin on console0"
}
}
}
}
```

•

# Additional References

## Related Documents

Related Topic	Document Title
Example configurations of telemetry deployment for VXLAN EVPN.	<a href="#">Telemetry Deployment for VXLAN EVPN Solution</a>





## APPENDIX **A**

# Streaming Telemetry Sources

---

- [About Streaming Telemetry, on page 383](#)
- [Guidelines and Limitations, on page 383](#)
- [Data Available for Telemetry, on page 383](#)

## About Streaming Telemetry

The streaming telemetry feature of Cisco Nexus 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 guideline and limitations for streaming telemetry:

- The telemetry feature is available in Cisco Nexus switches.
- Switches with less than 8 GB of memory do not support telemetry. The Cisco Nexus 3500-XL switch has 16 GB of memory and therefore supports 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.





## INDEX

### B

- Bash [5, 7](#)
  - accessing [5](#)
  - examples [7](#)
  - feature bash-shell [5](#)
- Bourne-Again SHell, *See* Bash

### F

- feature grpc [254](#)

### G

- grpc certificate [254–255](#)
- grpc gnmi max-concurrent-call [254–255](#)
- grpc port [254–255](#)

### N

- NX-API [129–131, 134, 136, 146, 161, 165, 170](#)
  - CLI [131](#)
  - cookie [130](#)
  - management commands [134](#)
  - message format [130](#)
  - request elements [136](#)
  - response codes [161](#)
  - response elements [146](#)
  - security [130](#)
  - transport [130](#)
  - user interface [165, 170](#)

### P

- Python [73–77, 79–81](#)
  - about API [73](#)
  - Cisco package [74](#)
  - CLI command API [75](#)
  - display format examples [77](#)
  - embedded event manager (EEM) [80](#)
  - invoking [76](#)
  - non-interactive [79](#)
  - NX-OS network interfaces [81](#)
  - NX-OS security [81](#)
  - scripts [73](#)

### T

- tcl [85–87, 90](#)
  - cli commands [86](#)
  - command separation [86](#)
  - history [86](#)
  - no interactive help [85](#)
  - options [87](#)
  - references [90](#)
  - sandbox [87](#)
  - security [87](#)
  - tab completion [86](#)
  - telquit command [87](#)
  - variables [87](#)
- Tool Command Language, *See* tcl

