Cisco Nexus 3600 NX-OS Programmability Guide, Release 9.3(x)

First Published: 2019-07-20

Last Modified: 2020-08-20

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

CONTENTS

PREFACE **Preface** xvii

Audience xvii

Document Conventions xvii

Related Documentation for Cisco Nexus 3600 Platform Switches xviii

Documentation Feedback xviii

Communications, Services, and Additional Information xviii

CHAPTER 1 **New and Changed Information** 1

New and Changed Information 1

CHAPTER 2 **Overview** 3

Programmability Overview 3

Licensing Requirements 4

Supported Platforms 4

Standard Network Manageability Features 4

Advanced Automation Feature 4

Power On Auto Provisioning Support 4

Programmability Support 5

NX-API Support 5

Python Scripting 5

Bash 5

Perl Modules 5

PART I **Shells and Scripting** 7

CHAPTER 3 **Bash** 9

About Bash 9
Guidelines and Limitations 9
Accessing Bash 9
Escalate Privileges to Root 11
Examples of Bash Commands 12
    Displaying System Statistics 12
    Running Bash from CLI 13
    Running Python from Bash 13
Managing RPMs 14
    Installing RPMs from Bash 14
    Upgrading RPMs 15
    Downgrading an RPM 15
    Erasing an RPM 15
Persistently Daemonizing an SDK- or ISO-built Third Party Process 16
Persistently Starting Your Application from the Native Bash Shell 17
Copy Through Kstack 17
An Example Application in the Native Bash Shell 18

CHAPTER 4 **Guest Shell** 21
About the Guest Shell 21
Guidelines and Limitations 22
Accessing the Guest Shell 26
Resources Used for the Guest Shell 27
Capabilities in the Guestshell 27
    NX-OS CLI in the Guest Shell 28
    Network Access in Guest Shell 28
    Access to Bootflash in Guest Shell 30
    Python in Guest Shell 31
    Python 3 in Guest Shell versions up to 2.10 (CentOS 7) 31
    Installing RPMs in the Guest Shell 34
Security Posture for Guest Shell 35
    Kernel Vulnerability Patches 35
    ASLR and X-Space Support 36
    Namespace Isolation 36

Root-User Restrictions 37
Resource Management 38
Guest File System Access Restrictions 38
Managing the Guest Shell 38
Disabling the Guest Shell 42
Destroying the Guest Shell 43
Enabling the Guest Shell 43
Replicating the Guest Shell 44
Exporting Guest Shell rootfs 44
Importing Guest Shell rootfs 45
Importing YAML File 46
show guestshell Command 49
Verifying Virtual Service and Guest Shell Information 50
Persistently Starting Your Application From the Guest Shell 51
Procedure for Persistently Starting Your Application from the Guest Shell 52
An Example Application in the Guest Shell 52
Troubleshooting Guest Shell Issues 53

CHAPTER 5 **Python API** 55
About the Python API 55
Using Python 55
Cisco Python Package 55
Using the CLI Command APIs 57
Invoking the Python Interpreter from the CLI 58
Display Formats 59
Non-Interactive Python 60
Running Scripts with Embedded Event Manager 61
Python Integration with Cisco NX-OS Network Interfaces 62
Cisco NX-OS Security with Python 62
Examples of Security and User Authority 63
Example of Running Script with Scheduler 64

CHAPTER 6 **Scripting with Tcl** 67
About Tcl 67

Guidelines and Limitations 67
Tclsh Command Help 67
Tclsh Command History 68
Tclsh Tab Completion 68
Tclsh CLI Command 68
Tclsh Command Separation 69
Tcl Variables 69
Tclquit 69
Tclsh Security 69
Running the Tclsh Command 70
Navigating Cisco NX-OS Modes from the Tclsh Command 71
Tcl References 72

CHAPTER 7 **iPXE** 73
About iPXE 73
Netboot Requirements 74
Guidelines and Limitations 74
Notes for iPXE 74
Boot Mode Configuration 82
Verifying the Boot Order Configuration 84

CHAPTER 8 **Kernel Stack** 85
About Kernel Stack 85
Guidelines and Limitations 85
Changing the Port Range 86

PART II **Applications** 89

CHAPTER 9 **Third-Party Applications** 91
About Third-Party Applications 91
Installing Signed Third-Party RPMs by Importing Keys Automatically 91
Installing Signed RPM 93
Checking a Signed RPM 93
Installing Signed RPMs by Manually Importing Key 94

Installing Signed Third-Party RPMs by Importing Keys Automatically 96

Adding Signed RPM into Repo 98

Persistent Third-Party RPMs 98

Installing RPM from VSH 99

Package Addition 99

Package Activation 100

Deactivating Packages 101

Removing Packages 101

Displaying Installed Packages 101

Displaying Detail Logs 102

Upgrading a Package 102

Downgrading a Package 102

Third-Party Applications 103

NX-OS 103

collectd 103

Ganglia 103

Iperf 103

LLDP 104

Nagios 104

OpenSSH 104

Quagga 104

Splunk 104

tcollector 104

tcpdump 105

Tshark 105

CHAPTER 10 **Ansible** 107

Prerequisites 107

About Ansible 107

Cisco Ansible Module 107

CHAPTER 11 **Puppet Agent** 109

About Puppet 109

Prerequisites 109

Puppet Agent NX-OS Environment 110

ciscopuppet Module 110

CHAPTER 12 **Using Chef Client with Cisco NX-OS** 113

About Chef 113

Prerequisites 113

Chef Client NX-OS Environment 114

cisco-cookbook 114

CHAPTER 13 **Nexus Application Development - ISO** 117

About ISO 117

Installing the ISO 117

Using the ISO to Build Applications 118

Using RPM to Package an Application 119

CHAPTER 14 **Nexus Application Development - SDK** 121

About the Cisco SDK 121

Installing the SDK 121

Procedure for Installation and Environment Initialization 122

Using the SDK to Build Applications 123

Using RPM to Package an Application 124

Creating an RPM Build Environment 125

Using General RPM Build Procedure 125

Example to Build RPM for collectd with No Optional Plug-Ins 126

Example to Build RPM for collectd with Optional Curl Plug-In 127

CHAPTER 15 **NX-SDK** 129

About the NX-SDK 129

Considerations for Go Bindings 130

About On-Box (Local) Applications 130

Default Docker Images 130

Guidelines and Limitations for NX-SDK 131

About NX-SDK 2.0 131

About NX-SDK 2.5 132

About Remote Applications 132

NX-SDK Security 133

Security Profiles for NX SDK 2.0 133

CHAPTER 16 **Using Docker with Cisco NX-OS** 135

About Docker with Cisco NX-OS 135

Guidelines and Limitations 135

Prerequisites for Setting Up Docker Containers Within Cisco NX-OS 136

Starting the Docker Daemon 136

Configure Docker to Start Automatically 137

Starting Docker Containers: Host Networking Model 138

Starting Docker Containers: Bridged Networking Model 139

Mounting the bootflash and volatile Partitions in the Docker Container 140

Enabling Docker Daemon Persistence on Enhanced ISSU Switchover 140

Resizing the Docker Storage Backend 141

Stopping the Docker Daemon 143

Docker Container Security 144

Securing Docker Containers With User namespace Isolation 144

Moving the cgroup Partition 145

Docker Troubleshooting 145

Docker Fails to Start 146

Docker Fails to Start Due to Insufficient Storage 146

Failure to Pull Images from Docker Hub (509 Certificate Expiration Error Message) 147

Failure to Pull Images from Docker Hub (Client Timeout Error Message) 147

Docker Daemon or Containers Not Running On Switch Reload or Switchover 148

Resizing of Docker Storage Backend Fails 148

Docker Container Doesn't Receive Incoming Traffic On a Port 148

Unable to See Data Port And/Or Management Interfaces in Docker Container 149

General Troubleshooting Tips 149

PART III **NX-API** 151

CHAPTER 17 **NX-API CLI** 153

About NX-API CLI 153

Transport 153
Message Format 154
Security 154
Using NX-API CLI 154
Escalate Privileges to Root on NX-API 156
NX-API Management Commands 157
Working With Interactive Commands Using NX-API 161
NX-API Request Elements 161
NX-API Response Elements 165
Restricting Access to NX-API 166
Updating an iptable 166
Making an Iptable Persistent Across Reloads 168
Table of NX-API Response Codes 169
XML and JSON Supported Commands 171
About JSON (JavaScript Object Notation) 172
Examples of XML and JSON Output 172

CHAPTER 18 **NX-API REST** 181
About NX-API REST 181
DME Config Replace Through REST 181
About DME Full Config Replace Through REST Put 181
Guidelines and Limitations 182
Replacing the System-Level Configuration Through REST PUT 182
Replacing Feature-Level Config Through REST PUT 183
Replacing Property-Level Config Through REST POST 184
Troubleshooting Config Replace for REST PUT 184

CHAPTER 19 **NX-API Developer Sandbox** 187
NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2) 187
About the NX-API Developer Sandbox 187
Guidelines and Limitations 188
Configuring the Message Format and Command Type 188
Using the Developer Sandbox 190
Using the Developer Sandbox to Convert CLI Commands to Payloads 190

NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later **193**

About the NX-API Developer Sandbox **193**

Guidelines and Limitations **194**

Configuring the Message Format and Input Type **196**

Using the Developer Sandbox **199**

Using the Developer Sandbox to Convert CLI Commands to REST Payloads **200**

Using the Developer Sandbox to Convert from REST Payloads to CLI Commands **202**

Using the Developer Sandbox to Convert from RESTCONF to json or XML **207**

PART IV **Model-Driven Programmability** **211**

CHAPTER 20 **Managing Components** **213**

About the Component RPM Packages **213**

Preparing For Installation **215**

Downloading Components from the Cisco Artifactory **216**

Installing RPM Packages **217**

Installing the Programmable Interface Base And Common Model Component RPM Packages **217**

CHAPTER 21 **Converting CLI Commands to Network Configuration Format** **219**

Information About XMLIN **219**

Licensing Requirements for XMLIN **219**

Installing and Using the XMLIN Tool **220**

Converting Show Command Output to XML **220**

Configuration Examples for XMLIN **221**

CHAPTER 22 **gNMI - gRPC Network Management Interface** **225**

About gNMI **225**

gNMI RPC and SUBSCRIBE **226**

Guidelines and Limitations for gNMI **227**

Configuring gNMI **229**

Configuring Server Certificate **230**

Generating Key/Certificate Examples **231**

Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(2) and Earlier **231**

Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3(3) and Later **232**

Verifying gNMI 234

Clients 240

Sample DME Subscription - PROTO Encoding 240

Capabilities 241

About Capabilities 241

Guidelines and Limitations for Capabilities 242

Example Client Output for Capabilities 243

Get 245

About Get 245

Guidelines and Limitations for Get 245

Set 246

About Set 246

Guidelines and Limitations for Set 246

Subscribe 247

Guidelines and Limitations for Subscribe 247

gNMI Payload 248

Streaming Syslog 251

About Streaming Syslog for gNMI 251

Guidelines and Limitations for Streaming Syslog - gNMI 251

Syslog Native YANG Model 252

Subscribe Request Example 252

Sample PROTO Output 253

Sample JSON Output 256

Troubleshooting 257

Gathering TM-Trace Logs 257

Gathering MTX-Internal Logs 257

CHAPTER 23 **Model Driven Telemetry** 261

About Telemetry 261

Telemetry Components and Process 261

High Availability of the Telemetry Process 263

Licensing Requirements for Telemetry 263

Installing and Upgrading Telemetry 263

Guidelines and Limitations for Model Driven Telemetry 264

Configuring Telemetry Using the CLI 269
    Configuring Telemetry Using the NX-OS CLI 269
    Configuring Cadence for YANG Paths 274
    Configuration Examples for Telemetry Using the CLI 276
    Displaying Telemetry Configuration and Statistics 280
    Displaying Telemetry Log and Trace Information 286
Configuring Telemetry Using the NX-API 288
    Configuring Telemetry Using the NX-API 288
    Configuration Example for Telemetry Using the NX-API 297
    Telemetry Model in the DME 300
Telemetry Path Labels 302
    About Telemetry Path Labels 302
    Polling for Data or Receiving Events 302
    Guidelines and Limitations for Path Labels 303
    Configuring the Interface Path to Poll for Data or Events 303
    Configuring the Interface Path for Non-Zero Counters 305
    Configuring the Interface Path for Operational Speeds 306
    Configuring the Interface Path with Multiple Queries 307
    Configuring the Environment Path to Poll for Data or Events 309
    Configuring the Resources Path for Poll for Events or Data 310
    Configuring the VXLAN Path to Poll for Events or Data 312
    Verifying the Path Label Configuration 313
    Displaying Path Label Information 314
Native Data Source Paths 316
    About Native Data Source Paths 316
    Telemetry Data Streamed for Native Data Source Paths 317
    Guidelines and Limitations for Native Data Source Path 318
    Configuring the Native Data Source Path for Routing Information 319
    Configuring the Native Data Source Path for MAC Information 320
    Configuring the Native Data Source Path for All MAC Information 322
    Configuring the Native Data Path for IP Adjacencies 324
    Displaying Native Data Source Path Information 326
Streaming Syslog 327
    About Streaming Syslog for Telemetry 327

Configuring the Native Data Source Path for Routing Information 327
Telemetry Data Streamed for Syslog Path 329
Sample JSON Output 330
Sample KVGPB Output 331
Additional References 333
Related Documents 333

PART V XML Management Interface 335

CHAPTER 24 XML Management Interface 337
About the XML Management Interface 337
About the XML Management Interface 337
NETCONF Layers 337
SSH xmlagent 338
Licensing Requirements for the XML Management Interface 338
Prerequisites to Using the XML Management Interface 339
Using the XML Management Interface 339
Configuring SSH and the XML Server Options 339
Starting an SSH Session 339
Sending the Hello Message 340
Obtaining the XSD Files 340
Sending an XML Document to the XML Server 341
Creating NETCONF XML Instances 341
RPC Request Tag rpc 342
NETCONF Operations Tags 343
Device Tags 344
Extended NETCONF Operations 346
NETCONF Replies 349
RPC Response Tag 350
Interpreting Tags Encapsulated in the Data Tag 350
Information About Example XML Instances 351
Example XML Instances 351
NETCONF Close Session Instance 351
NETCONF Kill-session Instance 352

NETCONF copy-config Instance 352

NETCONF edit-config Instance 352

NETCONF get-config Instance 354

NETCONF Lock Instance 354

NETCONF unlock Instance 355

NETCONF Commit Instance - Candidate Configuration Capability 356

NETCONF Confirmed-commit Instance 356

NETCONF rollback-on-error Instance 356

NETCONF validate Capability Instance 357

Additional References 357

Preface

This preface includes the following sections:

- Audience, on page xvii
- Document Conventions, on page xvii
- Related Documentation for Cisco Nexus 3600 Platform Switches, on page xviii
- Documentation Feedback, on page xviii
- Communications, Services, and Additional Information, on page xviii

Audience

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

Document Conventions

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
Italic	Italic text indicates arguments for which 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
`variable`	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

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

Related Documentation for Cisco Nexus 3600 Platform Switches

The entire Cisco Nexus 3600 platform switch documentation set is available at the following URL:

http://www.cisco.com/c/en/us/support/switches/nexus-3000-series-switches/tsd-products-support-series-home.html

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to nexus3k-docfeedback@cisco.com. We appreciate your feedback.

Communications, Services, and Additional Information

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

Cisco Bug Search Tool

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

CHAPTER 1

New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 3600 Series NX-OS Programmability Guide, 9.3(x)*.

• New and Changed Information, on page 1

New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 3600 Series NX-OS Programmability Guide, Release 9.3(x)*.

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
gNMI Get/Set	Add the Get and Set RPCs	9.3(5)	gNMI - gRPC Network Management Interface, on page 225
Python 3 on NX-OS	Python 3 support added.	9.3(5)	Python API, on page 55
YANG Support for Multiple Keys	Added configuration for a single telemetry DME stream.	9.3(5)	Model Driven Telemetry, on page 261
NX-SDK	Added support for NX-SDK 2.0, which includes support for remote (off-box) applications and enhanced security.	9.3(1)	NX-SDK, on page 129
NX-OS **copy** commands	Enabled speed and operability of the switch's **copy** commands	9.3(1)	Copy Through Kstack, on page 17
NX-OS **show** commands	Added JSON Native and JSON Native Pretty support for NX-OS **show** commands	9.3(1)	XML and JSON Supported Commands, on page 171

Feature	Description	Changed in Release	Where Documented
NX-API Chunking	Enhanced messages and chunking functionality	9.3(1)	NX-API CLI, on page 153
REST API	Added REST support for replacing the switch configuration at the tree, subtree, and leaf level of the DME.	9.3(1)	NX-API REST, on page 181
No updates since Cisco NX-OS Release 9.2(x)	First 9.3(x) release	Not applicable	Not applicable

CHAPTER 2

Overview

- Programmability Overview, on page 3
- Licensing Requirements, on page 4
- Supported Platforms, on page 4
- Standard Network Manageability Features, on page 4
- Advanced Automation Feature, on page 4
- Programmability Support, on page 5

Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 3600 platform switches is as follows:

- **Resilient**

  Provides critical business-class availability.

- **Modular**

  Has extensions that accommodate business needs.

- **Highly Programmatic**

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

- **Secure**

  Protects and preserves data and operations.

- **Flexible**

  Integrates and enables new technologies.

- **Scalable**

  Accommodates and grows with the business and its requirements.

- **Easy to use**

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

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

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

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

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.

Standard Network Manageability Features

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

Advanced Automation Feature

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

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

- Power On Auto Provisioning (POAP) support
- Chef and Puppet integration
- OpenStack integration
- OpenDayLight integration and OpenFlow support

Power On Auto Provisioning Support

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

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

Programmability Support

Cisco NX-OS software on switches support several capabilities to aid programmability.

NX-API Support

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

Python Scripting

Cisco NX-OS supports Python v2.7.5 in both interactive and noninteractive (script) modes.

Beginning in Cisco NX-OS Release 9.3(5), Python 3 is also supported.

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

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

Bash

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

Perl Modules

In order to support additional applications, the following Perl modules have been added:

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- Class/Struct.pm

- Data/Dumper.pm
- DynaLoader.pm
- Exporter/Heavy.pm
- FileHandle.pm
- File/Basename.pm
- File/Glob.pm
- File/Spec.pm
- File/Spec/Unix.pm
- File/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm
- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- Time/HiRes.pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so

PART I

Shells and Scripting

- Bash, on page 9
- Guest Shell, on page 21
- Python API, on page 55
- Scripting with Tcl, on page 67
- iPXE, on page 73
- Kernel Stack, on page 85

CHAPTER 3

Bash

- About Bash, on page 9
- Guidelines and Limitations, on page 9
- Accessing Bash, on page 9
- Escalate Privileges to Root, on page 11
- Examples of Bash Commands, on page 12
- Managing RPMs, on page 14
- Persistently Daemonizing an SDK- or ISO-built Third Party Process, on page 16
- Persistently Starting Your Application from the Native Bash Shell, on page 17
- Copy Through Kstack, on page 17
- An Example Application in the Native Bash Shell, on page 18

About Bash

In addition to the Cisco NX-OS CLI, 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.

Guidelines and Limitations

The Bash shell has the following guidelines and limitations:

- The binaries located in the `/isan` folder are meant to be run in an environment which is setup differently from that of the shell entered from the **run bash** command. It is advisable not to use these binaries from the Bash shell as the behavior within this environment is not predictable.

Accessing Bash

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

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

```
switch# show role name dev-ops
```

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

switch# show role name network-admin

Role: network-admin
  Description: Predefined network admin role has access to all commands
  on the switch
  -------------------------------------------------------------------
  Rule    Perm    Type        Scope               Entity
  -------------------------------------------------------------------
  1       permit  read-write
switch#
```

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

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

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

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

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

switch# run bash?
  bash  Linux-bash

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

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

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

run bash whoami

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

username foo shelltype bash

This command puts you directly into the Bash shell.

Escalate Privileges to Root

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

The following are guidelines for escalating privileges:

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

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

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

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

The example below demonstrates with network administrator user MyUser with a default shelltype Bash using **sudo** to pass configuration commands to the NX-OS:

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

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

The example below demonstrates with network administrator user MyUser with default shelltype Bash entering the NX-OS and then running Bash on the NX-OS:

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
Nexus 3600 software ("Nexus 3600 Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.

Any use or disclosure, in whole or in part, of the Nexus 3600 Software
or Documentation to any third party for any purposes is expressly
prohibited except as otherwise authorized by Cisco in writing.
The copyrights to certain works contained herein are owned by other
third parties and are used and distributed under license. Some parts
of this software may be covered under the GNU Public License or the
GNU Lesser General Public License. A copy of each such license is
available at
```

```
http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
***************************************************************************
*  Nexus 3600 is strictly limited to use for evaluation, demonstration      *
*  and NX-OS education. Any use or disclosure, in whole or in part of     *
*  the Nexus 3600 Software or Documentation to any third party for any      *
*  purposes is expressly prohibited except as otherwise authorized by     *
*  Cisco in writing.                                                      *
***************************************************************************
switch# run bash
bash-4.2$ vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface
 eth1/2 brief"

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

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

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

Examples of Bash Commands

This section contains examples of Bash commands and output.

Displaying System Statistics

The following example displays system statistics:

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

```
Mapped:            97832 kB
<\snip>
```

Running Bash from CLI

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

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

Running Python from Bash

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

```
switch# run bash
bash-4.2$ python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

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

version 6.1(2)I2(1)

interface Ethernet1/3
  vrf member myvrf

>>>
```

Managing RPMs

Installing RPMs from Bash

Procedure

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

Example

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

```
bash-4.2$ yum list installed | grep n9000
base-files.n9000                       3.0.14-r74.2                    installed
bfd.lib32_n9000                        1.0.0-r0                        installed
core.lib32_n9000                       1.0.0-r0                        installed
eigrp.lib32_n9000                      1.0.0-r0                        installed
eth.lib32_n9000                        1.0.0-r0                        installed
isis.lib32_n9000                       1.0.0-r0                        installed
lacp.lib32_n9000                       1.0.0-r0                        installed
linecard.lib32_n9000                   1.0.0-r0                        installed
lldp.lib32_n9000                       1.0.0-r0                        installed
ntp.lib32_n9000                        1.0.0-r0                        installed
nxos-ssh.lib32_n9000                   1.0.0-r0                        installed
ospf.lib32_n9000                       1.0.0-r0                        installed
perf-cisco.n9000_gdb                   3.12-r0                         installed
platform.lib32_n9000                   1.0.0-r0                        installed
shadow-securetty.n9000_gdb             4.1.4.3-r1                      installed
snmp.lib32_n9000                       1.0.0-r0                        installed
svi.lib32_n9000                        1.0.0-r0                        installed
sysvinit-inittab.n9000_gdb             2.88dsf-r14                     installed
tacacs.lib32_n9000                     1.0.0-r0                        installed
task-nxos-base.n9000_gdb               1.0-r0                          installed
tor.lib32_n9000                        1.0.0-r0                        installed
vtp.lib32_n9000                        1.0.0-r0                        installed
bash-4.2$ yum list available
bgp.lib32_n9000                        1.0.0-r0
bash-4.2$ sudo yum -y install bfd
```

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

Upgrading RPMs

Before you begin

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

Procedure

	Command or Action	Purpose
Step 1	**sudo yum -y upgrade** *rpm*	Upgrades an installed RPM.

Example

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

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

Downgrading an RPM

Procedure

	Command or Action	Purpose
Step 1	**sudo yum -y downgrade** *rpm*	Downgrades the RPM if any of the dnf repositories has a lower version of the RPM.

Example

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

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

Erasing an RPM

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

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

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

Procedure

	Command or Action	Purpose
Step 1	**sudo yum -y erase** *rpm*	Erases the RPM.

Example

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

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

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

Your application should have a startup bash script that gets installed in `/etc/init.d/`*`application_name`*. This startup bash script should have the following general format (for more information on this format, see http://linux.die.net/man/8/chkconfig).

```
#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
```

Persistently Starting Your Application from the Native Bash Shell

Procedure

Step 1 Install your application startup bash script that you created above into `/etc/init.d/`*application_name*

Step 2 Start your application with `/etc/init.d/`*application_name* start

Step 3 Enter **chkconfig --add** *application_name*

Step 4 Enter **chkconfig --level 3** *application_name* **on**

Run level 3 is the standard multi-user run level, and the level at which the switch normally runs.

Step 5 Verify that your application is scheduled to run on level 3 by running **chkconfig --list** *application_name* and confirm that level 3 is set to on

Step 6 Verify that your application is listed in `/etc/rc3.d`. You should see something like this, where there is an 'S' followed by a number, followed by your application name (tcollector in this example), and a link to your bash startup script in `../init.d/`*application_name*

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

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

bash-4.2#

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.

An Example Application in the Native Bash Shell

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

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

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

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

PIDFILE=/tmp/hello.pid

# See how we were called.
case "$1" in
start)
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
stop)
    kill -9 `cat $PIDFILE`
    RETVAL=$?
;;
status)
    ps -p `cat $PIDFILE`
    RETVAL=$?
;;
restart|force-reload|reload)
    kill -9 `cat $PIDFILE`
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

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

After reload

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

CHAPTER 4

Guest Shell

- About the Guest Shell, on page 21
- Guidelines and Limitations, on page 22
- Accessing the Guest Shell, on page 26
- Resources Used for the Guest Shell, on page 27
- Capabilities in the Guestshell, on page 27
- Security Posture for Guest Shell, on page 35
- Guest File System Access Restrictions , on page 38
- Managing the Guest Shell, on page 38
- Verifying Virtual Service and Guest Shell Information, on page 50
- Persistently Starting Your Application From the Guest Shell, on page 51
- Procedure for Persistently Starting Your Application from the Guest Shell, on page 52
- An Example Application in the Guest Shell, on page 52
- Troubleshooting Guest Shell Issues, on page 53

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 35 MB of RAM and 350 MB of bootflash when enabled. Use the **guestshell destroy** command to reclaim resources if the Guest Shell is not used.

Guidelines and Limitations

Common Guidelines Across All Releases

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

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

- If you are running a third-party DHCPD server in Guest Shell, 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 Guest Shell 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 Guest Shell and get a Bash prompt or run a command within the context of the Guest Shell. The command uses password-less SSH to an available port on the localhost in the default network namespace.
- The `sshd` utility can secure the pre-configured SSH access into the Guest Shell by listening on `localhost` to avoid connection attempts from outside the network. The `sshd` 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 nonprivileged user with host Bash access from being able to use the key to connect to the Guest Shell. Network-admin users may start another instance of sshd in the Guest Shell to allow remote access directly into the Guest Shell, but any user that logs into the Guest Shell is also given network-admin privilege.

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

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

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

- Installing the Cisco NX-OS software release on a fresh out-of-the-box switch will automatically enable the Guest Shell. Subsequent upgrades to the 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 updates CVEs only when CentOS makes them publicly 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 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 Guest Shell 1.0, you must 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 4 GB 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 NX-OS image that does not support the Guest Shell.

Pre-Configured SSHD Service

The Guest Shell starts an OpenSSH server upon boot up. The server listens on a randomly generated port on the localhost IP address interface 127.0.0.1 only. This provides the password-less connectivity into the Guest Shell from the NX-OS virtual-shell when the guestshell keyword is entered. If this server is killed or its configuration (residing in `/etc/ssh/sshd_config-cisco`) is altered, access to the Guest Shell from the NX-OS CLI might not work.

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

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

Note The Guest Shell sshd service for password-less access uses a randomized port 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/systm/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/sshd
   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 Guest Shell:

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

8. Save the configuration across multiple Guest Shell 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 user for SSH/SCP using the **ssh-keygen -t dsa** command.

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

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

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
```

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 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%	1/6%
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 dnf 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 dnf 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 **vrfinfo**, 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 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 <a href="http://www.cisco.com/">here</a>.</p>
</body></html>
[guestshell@guestshell bootflash]$
```

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

Example:

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

Within the Guest Shell, the network namespaces associated with the VRFs are what 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]# yum 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]# yum 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]# yum install -y rh-python36 | tail
warning: /var/cache/yum/x86_64/7/centos-sclo-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/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/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/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/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/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/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/df50cac98ea0d5b006c55a399c3bf1db9da7b5a24de7890bc9cfd5dd9e99/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/e8/74/6e4f91745020f967d09332bb2b8b9b10090957334692eb88ea4afe91b77f/urllib3-1.25.8-py2.py3-none-any.whl
 (125kB)
    100% |################################| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>
```

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

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

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

```
[admin@guestshell ~]$ sudo yum install -y rh-python35 | tail
Dependency Installed:
  rh-python35-python.x86_64 0:3.5.1-13.el7
  rh-python35-python-devel.x86_64 0:3.5.1-13.el7
  rh-python35-python-libs.x86_64 0:3.5.1-13.el7
  rh-python35-python-pip.noarch 0:7.1.0-2.el7
  rh-python35-python-setuptools.noarch 0:18.0.1-2.el7
  rh-python35-python-virtualenv.noarch 0:13.1.2-2.el7
  rh-python35-runtime.x86_64 0:2.0-2.el7
```

```
Complete!

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

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

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

Installing RPMs in the Guest Shell

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

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

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/.

Yum resolves the dependencies and installs all the required packages.

```
[guestshell@guestshell ~]$ sudo chvrf management yum -y install glibc.i686
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: bay.uchicago.edu
* extras: pubmirrors.dal.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-softokn-freebl-3.16.2.3-9.el7.i686.rpm | 187 kB 00:00:25
(2/2): glibc-2.17-78.el7.i686.rpm | 4.2 MB 00:00:30
------------------------------------------------------------------------------------------
Total 145 kB/s | 4.4 MB 00:00:30
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : nss-softokn-freebl-3.16.2.3-9.el7.i686 1/2
Installing : glibc-2.17-78.el7.i686 2/2
error: lua script failed: [string "%triggerin(glibc-common-2.17-78.el7.x86_64)"]:1: attempt
 to compare number with nil
Non-fatal "<"unknown">" scriptlet failure in rpm package glibc-2.17-78.el7.i686
Verifying : glibc-2.17-78.el7.i686 1/2
Verifying : nss-softokn-freebl-3.16.2.3-9.el7.i686 2/2

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

Dependency Installed:
nss-softokn-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 roofs** *size-in-MB* command is used to increase the size of the file system.

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

Security Posture for Guest Shell

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.

Kernel Vulnerability Patches

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

ASLR and X-Space Support

Cisco 3000 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 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 the Guest Shell are subject to restrictions imposed by reduced Linux capabilities. If your application must perform operations that require root privileges, restrict the use of the root account to the smallest set of operations that absolutely requires root access, and impose other controls such as a hard limit on the amount of time that the application can run in that mode.

The set of Linux capabilities that are dropped for root within the Guest Shell follow:

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

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

Resource Management

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

Guest File System Access Restrictions

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

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

Managing the Guest Shell

The following are commands to manage the Guest Shell:

Table 2: Guest Shell CLI Commands

Commands	Description
guestshell enable {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• When *guest shell OVA file* is specified: Installs and activates the Guest Shell using the OVA that is embedded in the system image. Installs and activates the Guest Shell using the specified software package (OVA file) or the embedded package from the system image (when no package is specified). Initially, Guest Shell packages are only available by being embedded in the system image. When the Guest Shell is already installed, this command enables the installed Guest Shell. Typically this is used after a **guestshell disable** command. • When *rootfs-file-URI* is specified: Imports a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.

Commands	Description
guestshell export rootfs package *destination-file-URI*	Exports a Guest Shell **rootfs** file to a local URI (bootflash, USB1, etc.).
guestshell disable	Shuts down and disables the Guest Shell.
guestshell upgrade {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• When *guest shell OVA file* is specified: Deactivates and upgrades the Guest Shell using the specified software package (OVA file) or the embedded package from the system image (if no package is specified). Initially Guest Shell packages are only available by being embedded in the system image. The current rootfs for the Guest Shell is replaced with the rootfs in the software package. The Guest Shell does not make use of secondary filesystems that persist across an upgrade. Without persistent secondary filesystems, a **guestshell destroy** command followed by a **guestshell enable** command could also be used to replace the rootfs. When an upgrade is successful, the Guest Shell is activated. You are prompted for a confirmation prior to carrying out the upgrade command. • When *rootfs-file-URI* is specified: Imports a Guest Shell **rootfs** file when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.

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

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

Note Administrative privileges are necessary to enable/disable and to gain access to the Guest Shell environment.

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

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        guestshe11.ova
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n) y

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

Note The Guest Shell is reactivated with the **guestshell enable** command.

Destroying the Guest Shell

The **guestshell destroy** command uninstalls the Guest Shell and its artifacts. The command does not remove the Guest Shell OVA.

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

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

switch# guestshell destroy

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

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

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

Note 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.

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 : 2.3(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

show virtual-service global
`switch# show virtual-service global`
 	
`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#`
	Displays the global state and limits for virtual services.

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

Command	Description

show guestshell detail
```
switch# show guestshell detail
Virtual service guestshell+ detail
  State                 : Activated
  Package information
    Name                : guestshell.ova
    Path                : /isan/bin/guestshell.ova
    Application
      Name              : GuestShell
      Installed version : 2.2(0.2)
      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
```
	Displays details about the guestshell package (such as version, signing resources, and devices).

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

- **About the Python API** , on page 55
- Using Python, on page 55

About the Python API

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

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

http://www.python.org/

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

The switches support Python v2.7.5 in both interactive and non-interactive (script) modes and is 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 PowerOn Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions. Python can also be accessed from the Bash shell.

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

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

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

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

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

NAME
    cisco

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

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

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

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 3: CLI Command APIs

API	Description

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

clid()
Example:
`json_string = clid (“cli-command”)`
	Returns JSON output for **cli-command**, if XML support exists for the command, otherwise an exception is thrown.
	Note This API can be useful when searching the output of show commands.

API	Description

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

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

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

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

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

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

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

Note Commands are separated with " *;* " as shown in the example. The 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.

```
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:

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.

Cisco NX-OS 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 %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 3600 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 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 3600
>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set global vrf in module cisco.vrf:
set global vrf(vrf)
Sets the global vrf. Any new sockets that are created (using socket.socket)
will automatically get set to this vrf (including sockets used by other
python libraries).
Arguments:
vrf: VRF name (string) or the VRF ID (int).
Returns: Nothing
>>>
```

Cisco NX-OS Security with Python

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

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

Examples of Security and User Authority

The following example shows how a privileged user runs commands:

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

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

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')
Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

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

The following is an example for a privileged user:

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

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

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue
```

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

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

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

The following example shows an RBAC configuration:

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

Example of Running Script with Scheduler

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

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

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

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

```
switch# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
switch(config)# feature scheduler
switch(config)# scheduler job name testplan
switch(config-job)# python bootflash:///scripts/testplan.py
switch(config-job)# exit
switch(config)# scheduler schedule name testplan
switch(config-schedule)# job name testplan
switch(config-schedule)# time start now repeat 0:0:4
Schedule starts from Mon Mar 14 16:40:03 2011
switch(config-schedule)# end
```

```
switch# term mon
2011 Mar 14 16:38:03 switch %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin on
10.19.68.246@pts/2
switch# show scheduler schedule
Schedule Name       : testplan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 4 Mins
Start Time          : Mon Mar 14 16:40:03 2011
Last Execution Time : Yet to be executed
-----------------------------------------------
     Job Name            Last Execution Status
-----------------------------------------------
    testplan                              -NA-
==============================================================================
switch#
switch# 2011 Mar 14 16:40:04 switch  %USER-1-SYSTEM_MSG: No user ran
/bootflash/scripts/testplan.py on : switch   - nxpython
2011 Mar 14 16:44:04 switch last message repeated 1 time
switch#
```

CHAPTER 6

Scripting with Tcl

- About Tcl, on page 67
- Running the Tclsh Command, on page 70
- Navigating Cisco NX-OS Modes from the Tclsh Command, on page 71
- Tcl References, on page 72

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.

Guidelines and Limitations

Following are guidelines and limitations for TCL scripting:

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

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.

Procedure

	Command or Action	Purpose

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

Example

The following example shows an interactive Tcl shell:

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

switch-tcl# exit
switch#
```

The following example shows how to run a Tcl script:

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

switch# tclsh bootflash:showmodule.tcl
Mod  Ports  Module-Type                         Model              Status
```

```
1    36     36p 40G Ethernet Module            N9k-X9636PQ        ok
Mod  Sw             Hw
Mod  MAC-Address(es)                         Serial-Num

switch#
```

Navigating Cisco NX-OS Modes from the Tclsh Command

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

Procedure

	Command or Action	Purpose

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

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

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

Example

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

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

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

Tcl References

The following titles are provided for your reference:

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

CHAPTER 7

iPXE

This chapter contains the following sections:

- About iPXE, on page 73
- Netboot Requirements, on page 74
- Guidelines and Limitations, on page 74
- Boot Mode Configuration, on page 82
- Verifying the Boot Order Configuration, on page 84

About iPXE

iPXE is an open source network boot firmware. iPXE is based on gPXE, which is an open-source PXE client firmware and bootloader derived from Etherboot. Standard PXE clients use TFTP to transfer data, whereas gPXE supports additional protocols.

Here is a list of additional features that iPXE provides over standard PXE:

- Boots from a web server via HTTP, iSCSI SAN, FCoE, etc.,
- Supports both IPv4 and IPv6,
- Netboot supports HTTP/TFTP, IPv4, and IPv6,
- Supports embedded scripts into the image or served by the HTTP/TFTP, etc., and
- Supports stateless address auto-configuration (SLAAC) and stateful IP auto-configuration variants for DHCPv6. iPXE supports boot URI and parameters for DHCPv6 options. This depends on IPv6 router advertisement.

In addition, we have disabled some of the existing features from iPXE for security reasons such as:

- Boot support for standard Linux image format such as bzImage+initramfs/initrd, or ISO, etc.,
- Unused network boot options such as FCoE, iSCSI SAN, Wireless, etc., and
- Loading of unsupported NBP (such as syslinux/pxelinux) because these might boot system images that are not properly code-signed.

Netboot Requirements

The primary requirements are:

- A DHCP server with proper configuration.
- A TFTP/HTTP server.
- Enough space on the device's bootflash because NX-OS downloads the image when the device is PXE booted.
- IPv4/IPv6 support—for better deployment flexibility

Guidelines and Limitations

PXE has the following configuration guidelines and limitations:

- While auto-booting through iPXE, there is a window of three seconds where you can enter **`Ctrl+B`** to exit out of the PXE boot. The system prompts you with the following options:

  ```
  Please choose a bootloader shell:
  1). GRUB shell
  2). PXE shell
  Enter your choice:
  ```

- HTTP image download vs. TFTP—TFTP is UDP based and it can be problematic if packet loss starts appearing. TCP is a window-based protocol and handles bandwidth sharing/losses better. As a result, TCP-based protocols support is more suitable given the sizes of the Cisco NX-OS images which are over 250 Mb.
- iPXE only allows/boots Cisco signed NBI images. Other standard image format support is disabled for security reasons.

Notes for iPXE

DHCP server installation

DHCP is not installed in the server by default. You can verify DHCP server installation with the **service dhcpd status** command.

```
[switch etc]# service dhcpd status
dhcpd: unrecognized service  /* indicates that dhcp server is not installed */
```

You can install DHCP with the **yum install dhcp** command.

Note Root credentials are required for installing the DHCP server.

```
[switch etc]# yum install dhcp
Repository base is listed more than once in the configuration
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package dhcp.x86_64 12:3.0.5-23.el5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved
==========================================================================================

 Package         Arch              Version                     Repository
Size
==========================================================================================
Installing:
 dhcp            x86_64            12:3.0.5-23.el5             workstation            883
 k

Transaction Summary
==========================================================================================
Install       1 Package(s)
Upgrade       0 Package(s)

Total download size: 883 k
Is this ok [y/N]: y
Downloading Packages:
dhcp-3.0.5-23.el5.x86_64.rpm                                          | 883 kB     00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : dhcp
1/1
Installed:
  dhcp.x86_64 12:3.0.5-23.el5

Complete!
[switch etc]#
```

Adding a configuration to the DHCP server

After the DHCP server is installed, the configuration file in located at **/etc/dhcpd.conf**.

The following is an example of the **dhcpd.conf** file.

```
-----------------------------------------------------------------------------------------------------------

 # Set the amount of time in seconds that  a client may keep the IP address
default-lease-time 300;
max-lease-time 7200;
one-lease-per-client true;

#Indicate the preferred interface that your DHCP server listens only to that interface and
 to no other . Preferred interface should be added to the DHCPDARGS variable
DHCPDARGS=eth0

#A subnet section is generated for each of the interfaces present on your Linux system
subnet 10.0.00.0 netmask 255.255.255.0 {

# The range of IP addresses the server will issue to DHCP enabled PC clients booting up on
 the network
```

```
  range 10.0.00.2 10.0.00.100;

#Address of the preferred inteface
  next-server 10.0.00.4;

#The default gateway to be used
  option routers 10.0.00.254;

#The file path where the  ipxe boot looks for the image
  filename = "http://10.0.00.4/pxe/dummy";
#  (http://10.0.00.4 points to the httpd service path mentioned in DocumentRoot variable
#   at /etc/httpd/conf/httpd.conf ) .
# By default it points to "DocumentRoot "/var/www/html"  (Refer the HTTP service section)

  option domain-name "cisco.com";
  option domain-name-servers 100.00.000.183;

host  Nexus {
        hardware ethernet e4:c7:22:bd:c4:f9;
        fixed-address 10.0.00.42;
        filename = "http://10.0.00.4/ipxe/nxos-image.bin";

host Nexus {
          hardware ethernet 64:f6:9d:07:52:f7;
          fixed-address 10.0.00.8;
          filename = "tftp://100.00.000.48/nxos-image.bin";
------------------------------------------------------------------------------------------------------------------------
```

Managing the DHCP service

Note After installing the DHCP service, you need to initiate the service.

- Verifying the DHCP service

```
[switch etc]# service dhcpd status
dhcpd is stopped
```

- Starting the DHCP service

```
[switch etc]#  service dhcpd start
Starting dhcpd:                                             [ok]
```

- Stopping the DHCP service

```
[switch etc]# service dhcpd stop
Stopping dhcpd:                                             [ok]
```

- Restarting the DHCP service

Note When the DHCP configuration file **/etc/dhcpd.conf** is updated, you need to restart the service.

```
[switch etc]# service dhcpd restart
Starting dhcpd:                                              [ok]
```

Managing the HTTP server

- HTTP server installation

```
[switch conf]# yum install httpd
```

- Starting the HTTP service

```
[switch conf]# service httpd start
Starting httpd: httpd: Could not reliably determine the server's fully qualified domain
 name,
using 100.00.000.127 for ServerName
                                                             [  OK  ]
```

- Stopping the HTTP service

```
[switch conf]# service httpd stop
Stopping httpd:                                              [  OK  ]
```

- Restarting the HTTP service

```
[switch conf]# service httpd restart
Stopping httpd:                                              [FAILED]
Starting httpd: httpd: Could not reliably determine the server's fully qualified domain
 name,
using 100.00.000.127 for ServerName
                                                             [  OK  ]
```

- Verifying the HTTP status

```
[switch conf]#  service httpd status
httpd (pid  23032) is running...
```

Note The HTTP configuration file is located at **/etc/httpd/conf/httpd.conf**.

Note

- DocumentRoot: The directory out of which you will serve your documents. By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations.
- DocumentRoot **/var/www/html**

  The DocumentRoot variable contains the path that represents the http://<ip_add> field in the **dhcpd.conf** file with the filename variable.

  The following is an example:

```
host  Nexus {
        hardware ethernet e4:c7:22:bd:c4:f9;
        fixed-address 10.0.00.42;
        filename =  "http://10.0.00.4/ipxe/nxos-image.bin";
```

  The filename path redirects to the location **/var/www/html/ipxe/nxos-image.bin**, where the ipxe bootup looks for the image .

- TFTP server installation

```
[switch conf]# yum install tftp
```

The TFTP configuration file located at **/etc/xinetd.d/tftp**.

The following is an example of a TFTP configuration file:

```
[switch xinetd.d]# cat tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot        # Indicates the tftp path
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

- Stopping the TFTP service

```
[switch xinetd.d]# chkconfig tftp off
```

- Starting the TFTP service

```
[switch xinetd.d]# chkconfig tftp on
```

Note When you change the TFTP configuration file, you need to restart the TFTP service.

```
host Nexus {
          hardware ethernet 64:f6:9d:07:52:f7;
          fixed-address 10.0.00.8;
          filename = "tftp://100.00.000.48/nxos-image.bin";
```

Note A prerequisite is that the nxos_image.bin has to be copied to **/tftpboot** shown in the above example TFTP path **/tftpboot**.

- iPXE using HTTP protocol

```
switch# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
  Internet Address is 10.0.00.42/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, medium is broadcast
  full-duplex, 100 Mb/s
  Auto-Negotiation is turned on
  Auto-mdix is turned off
  EtherType is 0x0000
  1 minute input rate 312 bits/sec, 0 packets/sec
  1 minute output rate 24 bits/sec, 0 packets/sec
  Rx
    5433 input packets 10 unicast packets 5368 multicast packets
    55 broadcast packets 405677 bytes
  Tx
    187 output packets 9 unicast packets 175 multicast packets
    3 broadcast packets 45869 bytes
switch#

switch# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# no boot nxos
switch(config)# boot order pxe bootflash
switch(config)# end

switch# copy running-config startup-config
[########################################] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.
switch# reload
This command will reboot the system. (y/n)?  [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x0000000380000000
 Relocated to memory
Time: 9/8/2017  1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revison        : 0x20
FPGA ID             : 0x1168153
FPGA Date           : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register  : 0x60ff
EventLog  Register1 : 0xc2004000
EventLog  Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type  1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok


Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6).................. ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
Filename: http://10.0.00.4/ipxe/nxos-image.bin
http://10.0.00.4/ipxe/nxos-image.bin... ok
http://10.0.00.4/ipxe/nxos_image.bin... 46%
Further device bootsup fine .
```

- iPXE using TFTP protocol

```
switch# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
  Internet Address is 10.0.00.8/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, medium is broadcast
  full-duplex, 100 Mb/s
  Auto-Negotiation is turned on
  Auto-mdix is turned off
  EtherType is 0x0000
  1 minute input rate 312 bits/sec, 0 packets/sec
  1 minute output rate 24 bits/sec, 0 packets/sec
  Rx
    5433 input packets 10 unicast packets 5368 multicast packets
    55 broadcast packets 405677 bytes
  Tx
    187 output packets 9 unicast packets 175 multicast packets
    3 broadcast packets 45869 bytes
switch#
switch# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms

switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# no boot nxos
switch(config)# boot order pxe bootflash
switch(config)# end

switch# copy running-config startup-config
[########################################] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.

switch# reload
This command will reboot the system. (y/n)?  [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x0000000380000000
 Relocated to memory
Time: 9/8/2017  1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revison        : 0x20
FPGA ID             : 0x1168153
FPGA Date           : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register  : 0x60ff
EventLog  Register1 : 0xc2004000
EventLog  Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type  1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
```

```
/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok


Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6).................. ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
filename: tftp://199.00.000.48/nxos-image.bin
tftp://199.00.000.48/nxos-image.bin... ok
tftp://199.00.000.48/nxos_image.bin... 26%



********************************************************************
```

- Interrupting the process

  Use crtl-B to interrupt the process and reach the iPXE shell.

  - The following is an example of booting an image residing on the PXE server using HTTP protocol:

    ```
    iPXE> dhcp
    Configuring (net6 e4:c7:22:bd:c4:a6)................ ok
    iPXE>boot  http://10.0.0.4/ipxe/nxos-image.bin
    ```

  - The following is an example of booting an image residing on the PXE server using TFTP protocol:

    ```
    iPXE> dhcp
    iPXE> boot tftp://199.00.00.48/nxos-image.bin
    ```

  Use **exit** to exit the iPXE shell.

Boot Mode Configuration

VSH CLI

```
switch# configure terminal
switch(conf)# boot order bootflash|pxe [bootflash|pxe]
switch(conf)# end
```

Note The keyword **bootflash** indicates it is Grub based booting.

For example, to do a PXE boot mode only, the configuration command is:

```
switch(conf)# boot order pxe
```

To boot Grub first, followed by PXE:

```
switch(conf)# boot order bootflash pxe
```

To boot PXE first, followed by Grub:

```
switch(conf)# boot order pxe bootflash
```

If you never use the **boot order** command, by default the boot order is Grub.

Note The following sections describe how you can toggle from Grub and iPXE.

Grub CLI

```
bootmode [-g|-p|-p2g|-g2p]
```

Keyword	Function
-g	Grub only
-p	PXE only
-p2g	PXE first, followed by Grub if PXE failed
-g2p	Grub first, followed by PXE if Grub failed

The Grub CLI is useful if you want to toggle the boot mode from the serial console without booting a full Cisco NX-OS image. It can also be used to get a box out of the continuous PXE boot state.

iPXE CLI

```
bootmode [-g|--grub] [-p|--pxe] [-a|--pxe2grub] [-b|--grub2pxe]
```

Keyword	Function
– – grub	Grub only
– – pxe	PXE only
– – pxe2grub	PXE first, followed by Grub if PXE failed
– – grub2pxe	Grub first, followed by PXE if Grub failed

The iPXE CLI is useful if you wish to toggle the boot mode from the serial console without booting a full Cisco NX-OS image. It can also be used to get a box out of continuous PXE boot state.

Verifying the Boot Order Configuration

To display boot order configuration information, enter the following command:

Command	Purpose
show boot order	Displays the current boot order from the running configuration and the boot order value on the next reload from the startup configuration.

CHAPTER 8

Kernel Stack

This chapter contains the following sections:

- About Kernel Stack, on page 85
- Guidelines and Limitations, on page 85
- Changing the Port Range, on page 86

About Kernel Stack

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

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

Kernel Stack has the following features:

Guidelines and Limitations

Using the Kernel Stack has the following guidelines and limitations:

- Guest Shell, other open containers, and the host Bash Shell use Kernel Stack (kstack).
- Open containers start in the host default namespace
    - Other network namespaces might be accessed by using the **setns** system call
    - The **nsenter** and **ip netns exec** utilities can be used to execute within the context of a different network namespace.
    - The PIDs and identify options for the **ip netns** command do not work without modification because of the file system device check. A **vrfinfo** utility is provided to give the network administrator the same information.
- Open containers may read the interface state from `/proc/net/dev` or use other normal Linux utilities such as **netstat** or **ifconfig** without modification. This provides counters for packets that have initiated / terminated on the switch.

- Open containers may use **ethtool –S** to get extended statistics from the net devices. This includes packets switched through the interface.
- Open containers may run packet capture applications like **tcpdump** to capture packets initiated from or terminated on the switch.
- There is no support for networking state changes (interface creation/deletion, IP address configuration, MTU change, etc.) from the Open containers
- IPv4 and IPv6 are supported
- Raw PF_PACKET is supported
- Well-known ports (0-15000) may only be used by one stack (Netstack or kstack) at a time, regardless of the network namespace.
- There is no IP connectivity between Netstack and kstack applications. This is a host limitation which also applies to open containers.
- Open containers are not allowed to send packets directly over an Ethernet out-of-band channel (EOBC) interface to communicate with the linecards or standby Sup.
- From within an open container, direct access to the EOBC interface used for internal communication with linecards or the standby supervisor. The host bash shell should be used if this access is needed.
- The management interface (mgmt0) is represented as eth1 in the kernel netdevices.
- Use of the VXLAN overlay interface (NVE x) is not supported for applications utilizing the kernel stack. NX-OS features, including CLI commands, are able to use this interface via netstack.

Changing the Port Range

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

- Kstack—15001 to 58000
- Netstack—58001 to 65535

Note Within this range 63536 to 65535 are reserved for NAT.

Procedure

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

Example

The following example sets the kstack port range:

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

What to do next

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

- You must reload the switch after entering the command.
- You must leave a minimum of 7000 ports unallocated which are used by Netstack.
- You must specify the *start-port* as 15001 or the *end-port* as 65535 to avoid holes in the port range.

PART II

Applications

- Third-Party Applications, on page 91
- Ansible, on page 107
- Puppet Agent, on page 109
- Using Chef Client with Cisco NX-OS, on page 113
- Nexus Application Development - ISO, on page 117
- Nexus Application Development - SDK, on page 121
- NX-SDK, on page 129
- Using Docker with Cisco NX-OS, on page 135

CHAPTER 9

Third-Party Applications

This chapter contains the following sections:

- About Third-Party Applications, on page 91
- Installing Signed Third-Party RPMs by Importing Keys Automatically, on page 91
- Installing Signed RPM, on page 93
- Persistent Third-Party RPMs, on page 98
- Installing RPM from VSH, on page 99
- Third-Party Applications, on page 103

About Third-Party Applications

The RPMs for the Third-Party Applications are available in the repository at https://devhub.cisco.com/artifactory/open-nxos/9.2.1/. These applications are installed in the native host by using the **yum** command in the Bash shell or through the NX-OS CLI.

When you enter the **yum install** *rpm* command, a Cisco **YUM** plug-in gets executed. This plug-in copies the RPM to a hidden location. On switch reload, the system reinstalls the RPM.

For configurations in `/etc`, a Linux process, **incrond**, monitors artifacts that are created in the directory and copies them to a hidden location, which gets copied back to `/etc`.

Installing Signed Third-Party RPMs by Importing Keys Automatically

Setup the yum repo to point to the keys and RPM.

```
root@switch# cat /etc/yum/repos.d/puppet.repo

[puppet]

name=Puppet RPM

baseurl=file:///bootflash/puppet

enabled=1

gpgcheck=1
```

```
gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs

metadata_expire=0

cost=500

bash-4.2# yum install puppet-enterprise

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages

groups-repo                              | 1.1 kB     00:00 ...

localdb                                  |  951 B     00:00 ...

patching                                 |  951 B     00:00 ...

puppet                                   |  951 B     00:00 ...

thirdparty                               |  951 B     00:00 ...

Setting up Install Process

Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed

--> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================

 Package             Arch     Version                            Repository     Size

==========================================================================================

Installing:

puppet-enterprise    x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos    puppet          14 M

Transaction Summary

==========================================================================================

Install       1 Package

Total download size: 14 M

Installed size: 46 M

Is this ok [y/N]: y

Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs

Importing GPG key 0x4BD6EC30:

 Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"
 From  : /bootflash/RPM-GPG-KEY-puppetlabs

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

Warning! Standby is not ready. This can cause RPM database inconsistency.

If you are certain that standby is not booting up right now, you may proceed.

Do you wish to continue?

Is this ok [y/N]: y

Warning: RPMDB altered outside of yum.

Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
                                                                        1/1

/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link

Installed:

puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos


Complete!
```

Installing Signed RPM

Checking a Signed RPM

Run the following command to check if a given RPM is signed or not.

```
Run, rpm -K rpm_file_name
```

Not a signed RPM

```
bash-4.2# rpm -K bgp-1.0.0-r0.lib32_n3600.rpm

bgp-1.0.0-r0.lib32_n3600.rpm: (sha1) dsa sha1 md5 OK
```

Signed RPM

```
bash-4.2#
rpm -K puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm: RSA sha1 MD5 NOT_OK
```

```
bash-4.2#
```

Signed third-party rpm requires public GPG key to be imported first before the package can be installed otherwise **yum** will throw the following error:

```
bash-4.2#
yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm -q

Setting up Install Process

warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30

Cannot open: puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm. Skipping.

Error: Nothing to do
```

Installing Signed RPMs by Manually Importing Key

- Copy the GPG keys to `/etc rootfs` so that they are persisted across reboots.

  ```
  bash-4.2# mkdir -p /etc/pki/rpm-gpg

  bash-4.2# cp -f RPM-GPG-KEY-puppetlabs /etc/pki/rpm-gpg/
  ```

- Import the keys using the below command

  ```
  bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs

  bash-4.2#

  bash-4.2# rpm -q gpg-pubkey

  gpg-pubkey-4bd6ec30-4c37bb40
  bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs

  bash-4.2#

  bash-4.2# rpm -q gpg-pubkey

  gpg-pubkey-4bd6ec30-4c37bb40
  ```

- Install the signed RPM with *yum* command

  ```
  bash-4.2#
  yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

  Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
  protect-packages

  groups-repo        | 1.1 kB     00:00 ...

         .
  localdb            |  951 B     00:00 ...


  patching           |  951 B     00:00 ...


  thirdparty         |  951 B     00:00 ...
  ```

```
Setting up Install Process

Examining puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm:
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64

Marking puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm to be installed

Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed


--> Finished Dependency ResolutionDependencies Resolved

=======================================================================================


 Package           Arch     Version                           Repository
Size

=======================================================================================


Installing:

puppet-enterprise  x86_64   3.7.1.rc2.6.g6cdc186-1.pe.nxos    /puppet-enterprise-
46 M
                                                              3.7.1.rc2.6.g6cdc186-1.
                                                              pe.nxos.x86_64


Transaction Summary

=======================================================================================


Install       1 Package

Total size: 46 M

Installed size: 46 M

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

  Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
                                                                            1/1


Installed:

  puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos
```

```
Complete!

bash-4.2#
```

Installing Signed Third-Party RPMs by Importing Keys Automatically

Setup the yum repo to point to the keys and RPM.

```
root@switch# cat /etc/yum/repos.d/puppet.repo

[puppet]

name=Puppet RPM

baseurl=file:///bootflash/puppet

enabled=1

gpgcheck=1

gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs

metadata_expire=0

cost=500

bash-4.2# yum install puppet-enterprise

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages

groups-repo                              | 1.1 kB     00:00 ...

localdb                                  |  951 B     00:00 ...

patching                                 |  951 B     00:00 ...

puppet                                   |  951 B     00:00 ...

thirdparty                               |  951 B     00:00 ...

Setting up Install Process

Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed

--> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================


 Package             Arch     Version                           Repository     Size

==========================================================================================
Installing:

puppet-enterprise    x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos    puppet          14 M

Transaction Summary

==========================================================================================


Install       1 Package

Total download size: 14 M

Installed size: 46 M

Is this ok [y/N]: y

Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs

Importing GPG key 0x4BD6EC30:

 Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"

 From  : /bootflash/RPM-GPG-KEY-puppetlabs

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

Warning! Standby is not ready. This can cause RPM database inconsistency.

If you are certain that standby is not booting up right now, you may proceed.

Do you wish to continue?

Is this ok [y/N]: y

Warning: RPMDB altered outside of yum.

Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
                                                                               1/1

/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link

Installed:

puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos


Complete!
```

Adding Signed RPM into Repo

Procedure

Step 1 **Copy signed RPM to repo directory**

Step 2 **Import the corresponding key for the create repo to succeed**

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm  RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# rpm --import RPM-GPG-KEY-puppetlabs
bash-4.2# createrepo .
1/1 - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
Saving Primary metadata
Saving file lists metadata
Saving other metadata
bash-4.2#
```

Without importing keys

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm  RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# createrepo .
warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30

Error opening package - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

Saving Primary metadata
Saving file lists metadata
Saving other metadata
```

Step 3 **Create repo config file under `/etc/yum/repos.d` pointing to this repo**

```
bash-4.2# cat /etc/yum/repos.d/puppet.repo
[puppet]
name=Puppet RPM
baseurl=file:///bootflash/puppet
enabled=1
gpgcheck=1
gpgkey=file:///bootflash/puppet/RPM-GPG-KEY-puppetlabs
#gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
metadata_expire=0
cost=500

bash-4.2# yum list available puppet-enterprise -q
Available Packages
puppet-enterprise.x86_64        3.7.1.rc2.6.g6cdc186-1.pe.nxos
                      puppet
bash-4.2#
```

Persistent Third-Party RPMs

The following is the logic behind persistent third-party RPMs:

- A local **yum** repository is dedicated to persistent third-party RPMs. The `/etc/yum/repos.d/thirdparty.repo` points to `/bootflash/.rpmstore/thirdparty`.
- Whenever you enter the **yum install third-party.rpm** command, a copy of the RPM is saved in `//bootflash/.rpmstore/thirdparty`.
- During a reboot, all the RPMs in the third-party repository are reinstalled on the switch.
- Any change in the `/etc` configuration files persists under `/bootflash/.rpmstore/config/etc` and they are replayed during boot on `/etc`.
- Any script that is created in the `/etc` directory persists across reloads. For example, a third-party service script that is created under `/etc/init.d/` brings up the apps during a reload.

Note The rules in iptables are not persistent across reboots when they are modified in a bash-shell.

To make the modified iptables persistent, see Making an Iptable Persistent Across Reloads, on page 168.

Installing RPM from VSH

Package Addition

NX-OS feature RPMs can also be installed by using the VSH CLIs.

Procedure

	Command or Action	Purpose
Step 1	**show install packages**	Displays the packages and versions that already exist.
Step 2	**install add ?**	Determine supported URIs.
Step 3	**install add** *rpm-packagename*	The **install add** command copies the package file to a local storage device or network server.

Example

The following example shows how to activate the Chef RPM:

```
switch# show install packages
switch# install add ?
WORD         Package name
bootflash:   Enter package uri
ftp:         Enter package uri
http:        Enter package uri
modflash:    Enter package uri
```

```
scp:        Enter package uri
sftp:       Enter package uri
tftp:       Enter package uri
usb1:       Enter package uri
usb2:       Enter package uri
volatile:   Enter package uri
switch# install add
bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5.x86_64.rpm
[####################] 100%
Install operation 314 completed successfully at Thu Aug  6 12:58:22 2015
```

What to do next

When you are ready to activate the package, go to Package Activation.

Note Adding and activating an RPM package can be accomplished in a single command:

```
switch#
install add bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5.x86_64.rpm
activate
```

Package Activation

Before you begin

The RPM has to have been previously added.

Procedure

	Command or Action	Purpose
Step 1	**show install inactive**	Displays the list of packages that were added and not activated.
Step 2	**install activate** *rpm-packagename*	Activates the package.

Example

The following example shows how to activate a package:

```
switch# show install inactive
Boot image:
        NXOS Image: bootflash:///yumcli6.bin

Inactive Packages:
        sysinfo-1.0.0-7.0.3.x86_64
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
Available Packages
chef.x86_64       12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5 thirdparty
eigrp.lib32_n3600 1.0.0-r0                                            groups-rep
o
```

```
sysinfo.x86_64    1.0.0-7.0.3                                        patching
switch# install activate chef-12.0-1.el5.x86_64.rpm
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

Deactivating Packages

Procedure

	Command or Action	Purpose
Step 1	**install deactivate** *package-name*	Deactivates the RPM package.

Example

The following example shows how to deactivate the Chef RPM package:

```
switch# install deactivate chef
```

Removing Packages

Before you begin

Deactivate the package before removing it. Only deactivated RPM packages can be removed.

Procedure

	Command or Action	Purpose
Step 1	**install remove** *package-name*	Removes the RPM package.

Example

The following example shows how to remove the Chef RPM package:

```
switch# install remove chef-12.0-1.el5.x86_64.rpm
```

Displaying Installed Packages

Procedure

	Command or Action	Purpose
Step 1	**show install packages**	Displays a list of the installed packages.

Example

The following example shows how to display a list of the installed packages:

```
switch# show install packages
```

Displaying Detail Logs

Procedure

	Command or Action	Purpose
Step 1	**show tech-support install**	Displays the detail logs.

Example

The following example shows how to display the detail logs:

```
switch# show tech-support install
```

Upgrading a Package

Procedure

	Command or Action	Purpose
Step 1	install add *package-name* activate upgrade	Upgrade a package.

Example

The following example show how to upgrade a package:

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate upgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

Downgrading a Package

Procedure

	Command or Action	Purpose
Step 1	install add *package-name* activate downgrade	Downgrade a package.

Example

The following example shows how to downgrade a package:

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate downgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

Third-Party Applications

NX-OS

For more information about NX-API REST API object model specifications, see https://developer.cisco.com/media/dme/index.html

collectd

collectd is a daemon that periodically collects system performance statistics and provides multiple means to store the values, such as RRD files. Those statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (that is, capacity planning).

For additional information, see https://collectd.org.

Ganglia

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

For additional information, see http://ganglia.info.

Iperf

Iperf was developed by NLANR/DAST to measure maximum TCP and UDP bandwidth performance. Iperf allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

For additional information, see http://sourceforge.net/projects/iperf/ or http://iperf.sourceforge.net.

LLDP

The link layer discover prototocol (LLDP) is an industry standard protocol designed to supplant proprietary link layer protocols such as EDP or CDP. The goal of LLDP is to provide an inter-vendor compatible mechanism to deliver link layer notifications to adjacent network devices.

For more information, see https://vincentbernat.github.io/lldpd/index.html.

Nagios

Nagios is open source software that monitors network services (through ICMP, SNMP, SSH, FTP, HTTP etc), host resources (CPU load, disk usage, system logs, etc.), and alert services for servers, switches, applications, and services through the Nagios remote plugin executor (NRPE) and through SSH or SSL tunnels.

For more information, see https://www.nagios.org/.

OpenSSH

OpenSSH is an open-source version of the SSH connectivity tools that encrypts all traffic (including passwords) to effectively eliminate eavesdropping, connection hijacking, and other attacks. OpenSSH provides secure tunneling capabilities and several authentication methods, and supports all SSH protocol versions.

For more information, see http://www.openssh.com.

Quagga

Quagga is a network routing software suite that implements various routing protocols. Quagga daemons are configured through a network accessible CLI called a "vty".

Note Only Quagga BGP has been validated.

For more information, see http://www.nongnu.org/quagga/.

Splunk

Splunk is a web based data collection, analysis, and monitoring tool that has a search, visualization and pre-packaged content for use-cases. The raw data is sent to the Splunk server using the Splunk Universal Forwarder. Universal Forwarders provide reliable, secure data collection from remote sources and forward that data into the Splunk Enterprise for indexing and consolidation. They can scale to tens of thousands of remote systems, collecting terabytes of data with minimal impact on performance.

For additional information, see http://www.splunk.com/en_us/download/universal-forwarder.html.

tcollector

tcollector is a client-side process that gathers data from local collectors and pushes the data to Open Time Series Database (OpenTSDB).

tcollector has the following features:

- Runs data collectors and collates the data,
- Manages connections to the time series database (TSD),
- Eliminates the need to embed TSD code in collectors,
- De-duplicates repeated values, and
- Handles wire protocol work.

For additional information, see http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html.

tcpdump

Tcpdump is a CLI application that prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight. It can also be run with the -w flag, which causes it to save the packet data to a file for later analysis, and/or with the -r flag, which causes it to read from a saved packet file rather than to read packets from a network interface. It can also be run with the -V flag, which causes it to read a list of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.

For more information, see http://www.tcpdump.org/manpages/tcpdump.1.html.

Tshark

TShark is a network protocol analyzer on the CLI. It lets you capture packet data from a live network, or read packets from a previously saved capture file, You can either print a decoded form of those packets to the standard output or write the packets to a file. TShark's native capture file format is the pcap format, which is also the format used by **tcpdump** and various other tools. Tshark can be used within the Guest Shell 2.1 after removing the cap_net_admin file capability.

```
setcap
 cap_net_raw=ep /sbin/dumpcap
```

Note This command must be run within the Guest Shell.

For more information, see https://www.wireshark.org/docs/man-pages/tshark.html.

CHAPTER 10

Ansible

- Prerequisites, on page 107
- About Ansible, on page 107
- Cisco Ansible Module, on page 107

Prerequisites

Go to 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	https://www.ansible.com/
Ansible Automation Solutions. Includes installation instructions, playbook instructions and examples, module lists, and so on.	https://docs.ansible.com/

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.	Configuration Management Tools

Ansible NX-OS playbook examples	Repo for ansible nxos playbooks
Ansible NX-OS network modules	nxos network modules

CHAPTER 11

Puppet Agent

This chapter includes the following sections:

- About Puppet, on page 109
- Prerequisites, on page 109
- Puppet Agent NX-OS Environment, on page 110
- ciscopuppet Module, on page 110

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	https://puppetlabs.com
Puppet Labs FAQ	https://puppet.com/blog/how-get-started-puppet-enterprise-faq/
Puppet Labs Documentation	https://puppet.com/docs

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)	https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md

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)	Puppet Forge
Resource Type Catalog	Cisco Puppet Resource Reference

ciscopuppet Module: Source Code Repository	Cisco Network Puppet Module
ciscopuppet Module: Setup & Usage	Cisco Puppet Module::README.md
Puppet Labs: Installing Modules	https://puppet.com/docs/puppet/7/modules_installing.html
Puppet NX-OS Manifest Examples	Cisco Network Puppet Module Examples
NX-OS developer landing page.	Configuration Management Tools

CHAPTER 12

Using Chef Client with Cisco NX-OS

This chapter includes the following sections:

- About Chef, on page 113
- Prerequisites, on page 113
- Chef Client NX-OS Environment, on page 114
- cisco-cookbook, on page 114

About Chef

Chef is an open-source software package that is developed by Chef Software, Inc. The software package 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 consists of one or more workstations, a single server, and every node that the chef-client has configured and is maintaining. 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 also can be used to install software packages, copy files, start services, and more.

The following references provide more information from Chef:

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

Prerequisites

The following are prerequisites for Chef:

- You must have a Cisco device and operating system software release that supports the installation:
    - Cisco Nexus 3600 platform switches
    - Cisco Nexus 3500 platform switches
    - Cisco Nexus 3100 platform switches
    - Cisco Nexus 3000 Series switch
    - Cisco NX-OS Release 7.0(3)I2(1) 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 a switch in the Guest Shell (the Linux container environment running CentOS). This software 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 NX-OS) install of chef-client is no longer supported.

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

Topic	Link
Chef Client: Installation and setup on Cisco Nexus platform (manual setup)	cisco-cookbook::README-install-agent.md
Chef Client: Installation and setup on a switch (automated installation using the Chef provisioner)	cisco-cookbook::README-chef-provisioning.md

cisco-cookbook

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

The cisco-cookbook can be found on Chef Supermarket.

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

Topic	Link
cisco-cookbook location	Chef Supermarket Cisco Cookbook

Topic	Link
Resource Type Catalog	Resource Catalog (by Technology)
cisco-cookbook: Source Code Repository	Cisco Network Chef Cookbook
cisco-cookbook: Setup and usage	Chef Cookbook Setup and Usage
Chef Supermarket	Chef Supermarket
Chef NX-OS Manifest Examples	Cisco Network Chef Cookbook Recipes

CHAPTER 13

Nexus Application Development - ISO

This chapter contains the following sections:

- About ISO, on page 117
- Installing the ISO, on page 117
- Using the ISO to Build Applications, on page 118
- Using RPM to Package an Application, on page 119

About ISO

The ISO image is a bootable Wind River 5 environment that includes the necessary tools, libraries, and headers to build and RPM-package third-party applications to run natively on a switch.

The content is not exhaustive, and it might be required that the user download and build any dependencies needed for any particular application.

Note Some applications are ready to be downloaded and used from the Cisco devhub website and do not require building.

Installing the ISO

The ISO image is available for download at: http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-I2-1/x86_64/satori-vm-intel-xeon-core.iso.

The ISO is intended to be installed as a virtual machine. Use instructions from your virtualization vendor to install the ISO.

Procedure

Step 1 (Optional) VMware-based installation.

The ISO image installation on a VMWare virtual machine requires the virtual disk to be configured as SATA and not SCSI.

Step 2 (Optional) QEMU-based installation.

Enter the following commands:

```
bash$ qemu-img create satori.img 10G
bash$ qemu-system-x86_64 -cdrom ./satori-vm-intel-xeon-core.iso -hda ./satori.img -m 8192
```

Once the ISO starts to boot, a menu is displayed. Choose the **Graphics Console Install** option. This installs to the virtual HD. Once the install is complete, the virtual machine must be rebooted.

What to do next

To login to the system, enter `root` as the login and `root` as the password.

Using the ISO to Build Applications Most of the build procedures that work with the SDK, and Linux in general, also apply to the ISO environment. However, there is no shell environment script to run. The default paths should be fine to use the toolsinstalled. The source code for applications needs to be obtained through the usual mechanisms such as a source tar file or git repository.

Build the source code:

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

bash$ make install
```

Using the ISO to Build Applications

Most of the build procedures that work with the SDK, and Linux in general, also apply to the ISO environment. However, there is no shell environment script to run. The default paths should be fine to use the tools installed. The source code for applications needs to be obtained through the usual mechanisms such as a source tar file or git repository.

Procedure

Build the source code.

a) **tar –xvzf example-lib.tgz**
b) **mkdir example-lib-install**
c) **cd example-lib/**
d) ./**configure** –**prefix**=*path_to_example-lib-install*
e) **make**
f) **make install**

The steps are normal Linux.

Example:

The following example shows how to build the source code:

```
bash$  tar -xvzf  example-lib.tgz
bash$  mkdir  example-lib-install
bash$  cd example-lib/
bash$  ./configure -prefix=<path_to_example-lib-install>
bash$  make
bach$ make install
```

Using RPM to Package an Application

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

Note **RPM and spec files**

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

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

CHAPTER 14

Nexus Application Development - SDK

This chapter contains the following sections:

- About the Cisco SDK, on page 121
- Installing the SDK, on page 121
- Procedure for Installation and Environment Initialization, on page 122
- Using the SDK to Build Applications, on page 123
- Using RPM to Package an Application, on page 124
- Creating an RPM Build Environment, on page 125
- Using General RPM Build Procedure, on page 125
- Example to Build RPM for collectd with No Optional Plug-Ins, on page 126
- Example to Build RPM for collectd with Optional Curl Plug-In, on page 127

About the Cisco SDK

The Cisco SDK is a development kit based on Yocto 1.2. It contains all of the tools needed to build applications for execution on a switch running the Cisco NX-OS Release 7.0(3)I2(1) and later. The basic components are the C cross-compiler, linker, libraries, and header files that are commonly used in many applications. The list is not exhaustive, and it might be required that the you download and build any dependencies needed for any particular application. Note that some applications are ready to be downloaded and used from the Cisco devhub website and do not require building. The SDK can be used to build RPM packages which may be directly installed on a switch.

Installing the SDK

The following lists the system requirements:

- The SDK can run on most modern 64-bit x86_64 Linux systems. It has been verified on CentOS 7 and Ubuntu 14.04. Install and run the SDK under the Bash shell.

- The SDK includes binaries for both 32-bit and 64-bit architectures, so it must be run on an x86_64 Linux system that also has 32-bit libraries installed.

Procedure

Check if the 32-bit libraries are installed:

Example:

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

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

- For CentOS 7:

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

- For Ubuntu 14.04:

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

Procedure for Installation and Environment Initialization

The SDK is available for download at: https://devhub.cisco.com/artifactory/open-nxos/10.0.1/nx-linux-x86_64-nxos-rootfs-n9k-sup-toolchain-1.1.0.sh

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

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

. environment-setup-corei7-64-nxos-linux
. environment-setup-corei7-32-nxosmllib32-linux

source environment-setup-corei7-64-nxos-linux
source environment-setup-corei7-32-nxosmllib32-linux
=============================
```

Use the **source environment-setup-x86_64-wrs-linux** command to add the SDK-specific paths to your shell environment. This must be done for each shell you intend to use with the SDK. This is the key to setting up the SDK in order to use the correct versions of the build tools and libraries.

Procedure

Step 1 Browse to the installation directory.

Step 2 Enter the following command at the Bash prompt:

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

Using the SDK to Build Applications

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

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

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

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

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

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

Example - Build Ganglia and its dependencies

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

libexpat

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

libapr

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

libconfuse

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

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

ganglia

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

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

Using RPM to Package an Application

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

Note **RPM and spec files**

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

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

Creating an RPM Build Environment

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

Procedure

Step 1 Create the directory structure:

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

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

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

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

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

Step 3 Refresh the RPM DB:

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

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

Using General RPM Build Procedure

General RPM Build procedure is as follows:

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app
tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

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

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

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

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

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

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

Download source tarball and extract spec file:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Add the following option:

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

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

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

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

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

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

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

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

CHAPTER 15

NX-SDK

This chapter contains the following topics:

- About the NX-SDK, on page 129
- About On-Box (Local) Applications, on page 130
- Default Docker Images, on page 130
- Guidelines and Limitations for NX-SDK, on page 131
- About NX-SDK 2.0 , on page 131
- About NX-SDK 2.5, on page 132
- About Remote Applications, on page 132
- NX-SDK Security, on page 133
- Security Profiles for NX SDK 2.0, on page 133

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 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 the Cisco Nexus switch in the Docker containers that 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 130
- 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 132.

Related Information

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

- Cisco DevNet NX-SDK. Click the `versions.md` link (https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md) to get information about features and details on each supported release.
- NX-SDK Readmes

As needed, Cisco adds 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 130.

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.

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 .

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.

About NX-SDK 2.5

Beginning with Cisco NX-OS Release 9.3(3), support is added for the Streaming Syslog feature.

For more information, see CiscoDevNet.

Table 4: Syslog Events

Features	Details
Syslog Events	• Ability for custom applications to register for Cisco NX-OS syslog events. • Refer to watchSyslog and postSyslogCb APIs in nx_trace.h for more details.

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.

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.

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 133.

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.
Throttle	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. 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.	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 16

Using Docker with Cisco NX-OS

This chapter contains the following topics:

- About Docker with Cisco NX-OS, on page 135
- Guidelines and Limitations, on page 135
- Prerequisites for Setting Up Docker Containers Within Cisco NX-OS, on page 136
- Starting the Docker Daemon, on page 136
- Configure Docker to Start Automatically, on page 137
- Starting Docker Containers: Host Networking Model, on page 138
- Starting Docker Containers: Bridged Networking Model, on page 139
- Mounting the bootflash and volatile Partitions in the Docker Container, on page 140
- Enabling Docker Daemon Persistence on Enhanced ISSU Switchover, on page 140
- Resizing the Docker Storage Backend, on page 141
- Stopping the Docker Daemon, on page 143
- Docker Container Security, on page 144
- Docker Troubleshooting, on page 145

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 1.13.1. 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 144 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.

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/userns-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 `--userns-remap=default` option to the `other_args` field in the `/etc/sysconfig/docker` file.

For example:

```
other_args="-debug=true --userns-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/userns-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
/ #
```

Docker Troubleshooting

These topics describe issues that can arise with Docker containers and provides possible resolutions.

Docker Fails to Start

Problem: Docker fails to start, showing an error message similar to the following:

```
switch# run bash
bash-4.3$ service docker start
Free bootflash: 39099 MB, total bootflash: 51771 MB
Carving docker bootflash storage: 2000 MB
2000+0 records in
2000+0 records out
2000000000 bytes (2.0 GB) copied, 22.3039 s, 89.7 MB/s
losetup: /dev/loop18: failed to set up loop device: Permission denied
mke2fs 1.42.9 (28-Dec-2013)
mkfs.ext4: Device size reported to be zero.  Invalid partition specified, or
        partition table wasn't reread after running fdisk, due to
        a modified partition being busy and in use.  You may need to reboot
        to re-read your partition table.

Failed to create docker volume
```

Possible Cause: You might be running Bash as an admin user instead of as a root user.

Solution: Determine if you are running Bash as an admin user instead of as a root user:

```
bash-4.3$ whoami
admin
```

Exit out of Bash and run Bash as root user:

```
bash-4.3$ exit
switch# run bash sudo su -
```

Docker Fails to Start Due to Insufficient Storage

Problem: Docker fails to start, showing an error message similar to the following, due to insufficient bootflash storage:

```
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 yum, 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.

PART III

NX-API

- NX-API CLI, on page 153
- NX-API REST, on page 181
- NX-API Developer Sandbox, on page 187

CHAPTER 17

NX-API CLI

- About NX-API CLI, on page 153
- Using NX-API CLI, on page 154
- XML and JSON Supported Commands, on page 171

About NX-API CLI

On switches, command-line interfaces (CLIs) are run only on the switch. NX-API CLI 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 switch. NX-API CLI supports **show** commands, configurations, and Linux Bash.

NX-API CLI supports JSON-RPC.

The NX-API CLI also supports JSON/CLI Execution in Cisco Nexus switches.

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 For the 7.x release, 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. In the 6.x release, all processes were killed when you ran the “no feature NXAPI” command, so this is a change in behavior in the 7.x release.

Message Format

Note

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

Security

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

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

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

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

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

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

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

Using NX-API CLI

The commands, command type, and output type for the 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.

Note For more details about NX-API response codes, see Table of NX-API Response Codes, on page 169.

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 CLI:

- Enable the management interface.

```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 192.0.20.123/24
switch(config)# vrf context managment
switch(config)# ip route 10.0.113.1/0 1.2.3.1
```

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

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

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

Request:

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

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

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

Request:

```
{
    "ins_api": {
        "version": "0.1",
        "type": "cli_show",
        "chunk": "0",
        "sid": "session1",
        "input": "show switchname",
        "output_format": "json"
    }
}
```

Response:

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

Escalate Privileges to Root on NX-API

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

The following are guidelines for escalating privileges:

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

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

First example:

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

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

```
</ins_api>
```

Second example:

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

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

NX-API Management Commands

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

Table 5: NX-API Management Commands

NX-API Management Command	Description
feature nxapi	Enables NX-API.
no feature nxapi	Disables NX-API.
nxapi {**http** | **https**} **port** *port*	Specifies a port.
no nxapi {**http** | **https**}	Disables HTTP/HTTPS.
show nxapi	Displays port and certificate information. **Note** The "**show nxapi**" command doesn't display certificate/config information for network-operator role.

NX-API Management Command	Description

nxapi certificate {httpscrt certfile | httpskey keyfile} *filename*	Specifies the upload of the following:
	• HTTPS certificate when httpscrt is specified.
	• HTTPS key when httpskey is specified.
	Example of HTTPS certificate:
	`nxapi certificate httpscrt certfile bootflash:cert.crt`
	Example of HTTPS key:
	`nxapi certificate httpskey keyfile bootflash:privkey.key`

nxapi certificatehttpskey keyfile *filename* **password** *passphrase*	Installs NX-API certificates with encrypted private keys: **Note** The passphrase for decrypting the encrypted private key is **pass123!**. Example: `nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!`
nxapi certificate enable	Enables a certificate.

NX-API Management Command	Description
nxapi certificate sudi	This CLI provides a secure way of authenticating to the device by using Secure Unique Device Identifier (SUDI). The SUDI based authentication in nginx will be used by the CISCO SUDI compliant controllers. SUDI is an IEEE 802.1AR-compliant secure device identity in an X.509v3 certificate which maintains the product identifier and serial number of Cisco devices. The identity is implemented at manufacturing and is chained to a publicly identifiable root certificate authority. **Note** When NX-API comes up with the SUDI certificate, it is not accessible by any third-party applications like browser, curl, and so on. **Note** "nxapi certificate sudi" will overwrite the custom certificate/key if configured, and there is no way to get the custom certificate/key back. **Note** "nxapi certificate sudi" and "nxapi certificate trustpoint" and "nxapi certificate enable" are mutually exclusive , and configuring one will delete the other configuration. **Note** NX-API do not support SUDI certificate-based client certificate authentication. If client certificate authentication is needed, then Identity certificate need to be used. **Note** As NX-API certificate CLI is not present in show run output, CR/Rollback case currently does not go back to the custom certificate once it is overwritten with "nxapi certificate sudi" options.
nxapi ssl-ciphers weak	Beginning with Cisco NX-OS Release 9.2(1), weak ciphers are disabled by default. Running this command changes the default behavior and enables the weak ciphers for NGINX. The **no** form of the command changes it to the default (by default, the weak ciphers are disabled).
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	Beginning with Cisco NX-OS Release 9.2(1), TLS1.0 is disabled by default. Running this command enables the TLS versions specified in the string, including the TLS1.0 that was disabled by default, if necessary. The **no** form of the command changes it to the default (by default, only TLS1.1 and TLS1.2 will be enabled).
nxapi use-vrf *vrf*	Specifies the default VRF, management VRF, or named VRF.

NX-API Management Command	Description
ip netns exec management iptables	Implements any access restrictions and can be run in management VRF. **Note** You must enable **feature bash-shell** and then run the command from Bash Shell. For more information on Bash Shell, see the chapter on Bash. *Iptables is a command-line firewall utility that uses policy chains to allow or block traffic and almost always comes pre-installed on any Linux distribution.* **Note** For more information about making iptables persistent across reloads when they are modified in a bash-shell, see Making an Iptable Persistent Across Reloads, on page 168.
nxapi idle-timeout <timeout>	Starting with Release 9.3(5), you can configure the amount of time before an idle NX-API session is invalidated. The time can be 1 - 1440 minutes. The default time is 10 minutes. Return to the default value by using the no form of the command: **no nxapi idle-timeout <timeout>**

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

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

Note You must configure the certificate and key before enabling the certificate.

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)#
```

The following is an example of how to install an encrypted NXAPI server certificate:

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!

switch(config)#nxapi certificate enable
switch(config)#
```

In some situations, you might get an error message saying that the key file is encrypted:

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
ERROR: Unable to load private key!
Check keyfile or provide pwd if key is encrypted, using 'nxapi certificate httpskey keyfile
 <keyfile> password <passphrase>'.
```

In this case, the passphrase of the encrypted key file must be specified using **nxapi certificatehttpskey keyfile** *filename* **password** *passphrase*.

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 password pass123!
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 device in XML format or JSON format. The HTTP header of the request must identify the content type of the request.

You use the NX-API elements that are listed in the following table to specify a CLI command:

Table 6: NX-API Request Elements for XML or JSON Format

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

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

NX-API Request Element	Description
chunk	Some **show** commands can return a large amount of output. For the NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for **show** commands. Enable or disable chunk with the following settings: 0 — Do not chunk output. 1 — Chunk output. **Note** • Only **show** commands support chunking. When a series of **show** commands are entered, only the first command is chunked and returned. • The output message format options are XML or JSON. • For the XML output message format , special characters, such as < or >, are converted to form a valid XML message (< is converted into &lt; > is converted into &gt). You can use XML SAX to parse the chunked output. • When the output message format is JSON, the chunks are concatenated to create a valid JSON object. **Note** When chunking is enabled, the maximum message size supported is currently 200MB of chunked output.
rollback	Valid only for configuration CLIs, not for show commands. Specifies the configuration rollback options. Specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in. **Note** The rollback element is available in the cli_conf mode when the input request format is XML or JSON.

NX-API Request Element	Description
sid	The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a *sid* to match the *sid* of the previous response message. NX-OS release 9.3(1) introduces the *sid* option `clear`. When a new chunk request is initiated with the *sid* set to `clear`, all current chunk requests are discarded or abandoned. When you receive response code `429: Max number of concurrent chunk request is 2`, use *sid* `clear` to abandon the current chunk requests. After using *sid* `clear`, subsequent response codes operate as usual per the rest of the request.
input	Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, **show** commands are cli_show message type and are not supported in cli_conf mode. **Note** Except for **bash**, multiple commands are separated with " ; ". (The ; must be surrounded with single blank characters.) For **bash**, multiple commands are separated with ";". (The ; is **not** surrounded with single blank characters.) The following are examples of multiple commands: cli_show: `show version ; show interface brief ; show vlan` cli_conf: `interface Eth4/1 ; no shut ; switchport` bash: `cd /bootflash;mkdir new_dir`
output_format	The available output message formats are the following: 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 7: NX-API Request Elements for JSON-RPC Format

NX-API Request Element	Description
jsonrpc	A string specifying the version of the JSON-RPC protocol. Version must be 2.0.

NX-API Request Element	Description
method	A string containing the name of the method to be invoked. NX-API supports either: • **cli**–show or configuration commands • **cli_ascii**–show or configuration commands; output without formatting • **cli_array**–only for show commands; similar to **cli**, but with **cli_array**, data is returned as a list of one element, or an array, within square brackets, [ ].
params	A structured value that holds the parameter values used during the invocation of a method. It must contain the following: • **cmd**–CLI command • **version**–NX-API request version identifier
rollback	Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores the failed CLI and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
id	An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the id parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, *id* : 1

NX-API Response Elements

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

Table 8: NX-API Response Elements

NX-API Response Element	Description
version	NX-API version.
type	Type of command to be executed.

NX-API Response Element	Description
sid	Session ID of the response. This element is valid only when the response message is chunked.
outputs	Tag that encloses all command outputs. When multiple commands are in cli_show or cli_show_ascii, each command output is enclosed by a single output tag. When the message type is cli_conf or bash, there is a single output tag for all the commands because cli_conf and bash commands require context.
output	Tag that encloses the output of a single command output. For cli_conf and bash message types, this element contains the outputs of all the commands.
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution. NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml).
msg	Error message associated with the returned error code.

Restricting Access to NX-API

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

Use ACLs to restrict HTTP or HTTPS access to a device only if you have not configured NXAPI to use a specific VRF. For information about configuring ACLs, see the *Cisco Nexus Series NX-OS Security Configuration Guide* for your switch family.

If you have configured a VRF for NX-API communication, however, ACLs will not restrict HTTP or HTTPS access. Instead, create a rule for an iptable. For more information about creating a rule, see Updating an iptable, on page 166.

Updating an iptable

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

Procedure

Step 1 To create a rule that blocks HTTP access:

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

Step 2 To create a rule that blocks HTTPS access:

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

Step 3 To verify the applied rules:

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

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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

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

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

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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

Step 5 To remove and verify previously applied rules:

This example removes the first rule from INPUT.

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

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

What to do next

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. To make the rules persistent, see Making an Iptable Persistent Across Reloads, on page 168.

Making an Iptable Persistent Across Reloads

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. This section explains how to make a modified iptable persistent across a reload.

Before you begin

You have modified an iptable.

Procedure

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

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

Step 2 Create the /etc/sys/iptables file where your iptables changes will be saved:

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

Step 3 Create a startup script called iptables_init in the /etc/init.d directory with the following set of commands:

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:

# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables

# Description:       sets config for iptables

#                    during boot time

### END INIT INFO


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
  start)
    start_script
```

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

Step 4 Set the appropriate permissions to the startup script:

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

Step 5 Set the iptables_int startup script to on with the chkconfig utility:

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

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

Table of NX-API Response Codes

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

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

Table 9: 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 `eoc` 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 **show** .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive show commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
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 `show` commands.
CHUNK_TIMEOUT	501	Timeout while generating chunk response.
CLI_CMD_NOT_SUPPORTED_ERR	501	CLI command not supported.
JSON_NOT_SUPPORTED_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_SUPPORTED_ERR	501	Output redirection is not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
PIPE_NOT_ALLOWED_IN_INPUT	501	Pipe is not allowed for this input type.
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_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

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.

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. JSON Pretty format, as well as JSON Native and JSON Pretty Native, is also supported.

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

CLI Execution

```
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#
```

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_oifl": "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_oifl>2</used_mcast_oifl>
           <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://w
ww.cisco.com/nxos:1.0:lldp">
 <nf:data>
  <show>
   <lldp>
    <timers>
     <__XML__OPT_Cmd_lldp_show_timers___readonly__>
      <__readonly__>
       <ttl>120</ttl>
       <reinit>2</reinit>
       <tx_interval>30</tx_interval>
       <tx_delay>2</tx_delay>
```

```
        <hold_mplier>4</hold_mplier>
        <notification_interval>5</notification_interval>
       </__readonly__>
      </__XML__OPT_Cmd_lldp_show_timers___readonly__>
     </timers>
    </lldp>
   </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

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_summar
y": "0", "acl_name": "acl-test1", "statistics": "enable", "frag_opt_permit_deny
": "permit-all", "TABLE_seqno": {"ROW_seqno": {"seqno": "10", "permitdeny": "pe
rmit", "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_t
os0_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,"m
ax_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cn
t":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","ins
tance_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":"PT0
S","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_pe
riod":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only"
:"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_d
ist":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":"PT
5S","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,"aso
paque_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_d
iscard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_num
ber":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","g
r_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":"PT
0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_h
old_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pac
e":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_norm
al":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_a
rea_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"
                                    },
                                    {
                                        "mask_len": "32",
                                        "count": "324"
                                    }
                                ]
                            }
                        }
                    }
                    }
                }
            }
        }
    }
switch-1#
```

CHAPTER 18

NX-API REST

This chapter contains the following sections:

- About NX-API REST, on page 181
- DME Config Replace Through REST, on page 181

About NX-API REST

NX-API REST

On switches, configuration is performed using command-line interfaces (CLIs) that run only on the switch. NX-API REST improves the accessibility of the switch configuration by providing HTTP/HTTPS APIs that:

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

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

NX-API REST uses HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/HTTPS POST body. The NX-API REST 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 resource usage exceeds the cgroup limitations, the Nginx process is restarted and restored.

For more information about the NX-API REST SDK, see https://developer.cisco.com/site/nx-api/documents/n3k-n9k-api-ref/.

DME Config Replace Through REST

About DME Full Config Replace Through REST Put

Beginning with Cisco NX-OS Release 9.3(1), Cisco NX-OS supports model-based full config replace through REST PUT operations. This method of replacing configurations uses the Cisco DME model.

The DME Full Config replace feature enables you to use the REST programmatic interface to replace the switch running configuration. The feature provides the following extra benefits:DME full config replace occurs through a PUT operation. All parts of the config tree (system-level, subtree, and leaf) support DME full config replace.

- Supports non-disruptive replacement of the switch configuration
- Supports automation
- Offers the ability to selectively modify features without affecting other features or their configs.
- Simplifies config changes and eliminates human error by enabling you to specify the final config outcome. The switch calculates the differences and pushes them to the affected parts of config tree.

Note Although not accomplished through a programmatic interface, you can also achieve a full config replace by using the **config replace config-file-name** Cisco NX-OS CLI command.

Guidelines and Limitations

The following are the guidelines and limitations for the DME full config replace feature:

- It is important for you to know the tree and know where you are applying the config replace. If you are using the Sandbox for the config replace operation, the Sandbox defaults to the subtree, so you might need to change the URI to target the correct node in the config tree.
- If you use the NX-OS Sandbox to Convert (for Replace), you must use the POST operation because of the presence of the `status: 'replaced'` attribute in the request. If you are using any other conversion option, you can use the PUT operation.
- If you use the REST PUT option for this feature on a subtree node, config replace operation is applied to the entire subtree. The target subtree node is correctly changed with the config replace data in the PUT, but be aware that leaf nodes of the subtree node are also affected by being set to default values.

  If you do not want the leaf nodes to be affected, do not use a PUT operation. Instead, you can use a POST operation with the `status:'replaced'` attribute.

  If you are applying the config replace to a leaf node, the PUT operation operates predictably.

Replacing the System-Level Configuration Through REST PUT

You can replace the entire configuration for the switch by sending a REST PUT from the management client.

Use the following procedure:

Procedure

Step 1 From the client, issue a REST PUT operation with the payload as the System level with the URL as `/api/mo/sys.json`.

The payload must be a valid config, and the config must be retrievable from the switch at any time by issuing a GET on `/api/mo/sys.json?rsp-subtree=full&rsp-prop-include=set-config-only`.

Step 2 Send a GET on the DN you used for the config replace by using `/api/mo/sys.json?rsp-subtree=full&rsp-prop-include=set-config-only`.

Step 3 (Optional) Compare the payload that you sent with the GET on the DN you replaced. The payload of the GET should be the same as the payload you sent.

Replacing Feature-Level Config Through REST PUT

Cisco DME supports replacing feature-level configurations through REST PUT operations. You can replace the configuration for specific features by sending a PUT at the feature level of the model.

Use the following procedure:

Procedure

Step 1 From the client, issue a REST PUT operation at the model object (MO) of the feature:

a) The Put must specify the URL from the top System level to the MO of the feature.

   For example, for a BGP `/api/mo/sys/bgp.json`

   The payload must be a valid config, and the config must be retrievable from the switch at any time by issuing a GET on the DN of the feature. For example, for BGP, `/api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only`.

b) The payload for the feature should start with the MO that you want to replace (for example, `bgp`).

For example:

```
{
            "bgpInst": {
              "attributes": {
                "asn": "100",
                "rn": "inst"
              },
              "children": [

              ... content removed for brevity ...

                {
                  "bgpDom": {
                    "attributes": {
                      "name": "vrf1",
                      "rn": "dom-vrf1"
                    },
                    "children": [
                      {
                        "bgpPeer": {
                          "attributes": {
                            "addr": "10.1.1.1",
                            "inheritContPeerCtrl": "",
                            "rn": "peer-[10.1.1.1]"
                          }
                        }
                      }
                    ]
                  }
                },
                {
                  "bgpDom": {
                    "attributes": {
                      "name": "default",
                      "rn": "dom-default",
                      "rtrId": "1.1.1.1"
                    }
                  }
                }
              ]
            }
          }
```

Step 2 Send a GET on the DN you used for the config replace by using `/api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only`.

Step 3 (Optional) Compare the payload that you sent with the GET on the DN you replaced. The payload of the GET should be the same as the payload you sent.

Replacing Property-Level Config Through REST POST

Cisco's DME model supports property-level config replace for CLI-based features through a REST POST operation. You can replace the config for the property of a feature through the NX-OS Sandbox by generating a request payload and sending it to the switch through a REST POST operation. For information about the NX-OS Sandbox, see NX-API Developer Sandbox, on page 187.

Procedure

Step 1 Connect to the switch through NX-OS Sandbox through HTTPS and provide your login credentials.

Step 2 In the work area, enter the CLI for the feature that you want to change.

Step 3 In the field below the work area, set the URI to the MO in the tree for the feature that you want to configure. This MO level is where you will send the Put request.

Step 4 For Method, select `NX-API (DME)`.

Step 5 For Input Type, select CLI.

Step 6 From the Convert drop-down list, select `Convert (for replace)` to generate the payload in the Request pane.

Step 7 Click the request using a **POST** operation to the switch..

Note Property-level config replace can fail if the config is a default config because the replace operation tries to delete all the children MOs and reset all properties to default.

Troubleshooting Config Replace for REST PUT

The following are steps to help troubleshoot if config replace through a REST Put operation is not successful.

Procedure

Step 1 Check if the request is valid.

The URL, operation, and payload should be valid. For example, if the URL is `api/mo/sys/foo.json` then the payload should start with `foo`

Step 2 Make sure the payload is valid and contains only the config properties which are:

- Successfully set
- Taken from a valid device config

To get only the config properties, use a GET that filters for `rsp-subtree=full&rsp-prop-include=set-config-only`

Step 3 To validate the payload, send it to the switch using a DME POST operation.

Step 4 Check the error to verify that it has the name of the MO and property.

Step 5 Validate the payload also has the name of the MO and property.

CHAPTER 19

NX-API Developer Sandbox

This chapter contains the following sections:

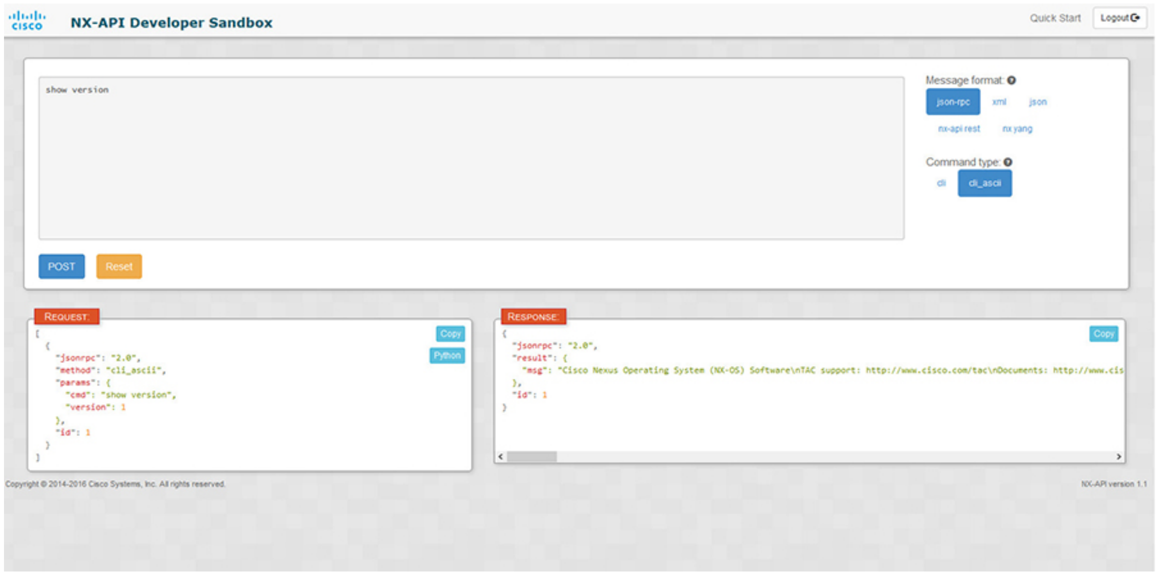
- NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2), on page 187
- NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later, on page 193

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.



```
show version
```

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli_ascii",
    "params": {
      "cmd": "show version",
      "version": 1
    },
    "id": 1
  }
]
```

```
{
  "jsonrpc": "2.0",
  "result": {
    "msg": "Cisco Nexus Operating System (NX-OS) Software\nTAC support: http://www.cisco.com/tac\nDocuments: http://www.cis
  },
  "id": 1
}
```

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 10: 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 jsonrpc.org.
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 Cisco Nexus NX-API References.
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 11: Command Types

Message format	Command type
json-rpc	• cli — show or configuration commands • cli-ascii — show or configuration commands, output without formatting
xml	• cli_show — show commands. If the command does not support XML output, an error message will be returned. • cli_show_ascii — show commands, output without formatting • cli_conf — configuration commands. Interactive configuration commands are not supported. • bash — bash commands. Most non-interactive bash commands are supported. **Note** The bash shell must be enabled in the switch.
json	• cli_show — show commands. If the command does not support XML output, an error message will be returned. • cli_show_ascii — show commands, output without formatting • cli_conf — configuration commands. Interactive configuration commands are not supported. • bash — bash commands. Most non-interactive bash commands are supported. **Note** The bash shell must be enabled in the switch.
nx-api rest	• cli — configuration commands • model — DN and corresponding payload.
nx yang	• json — JSON structure is used for payload • xml — XML structure is used for payload

Output Chunking

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

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

Using the Developer Sandbox

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.

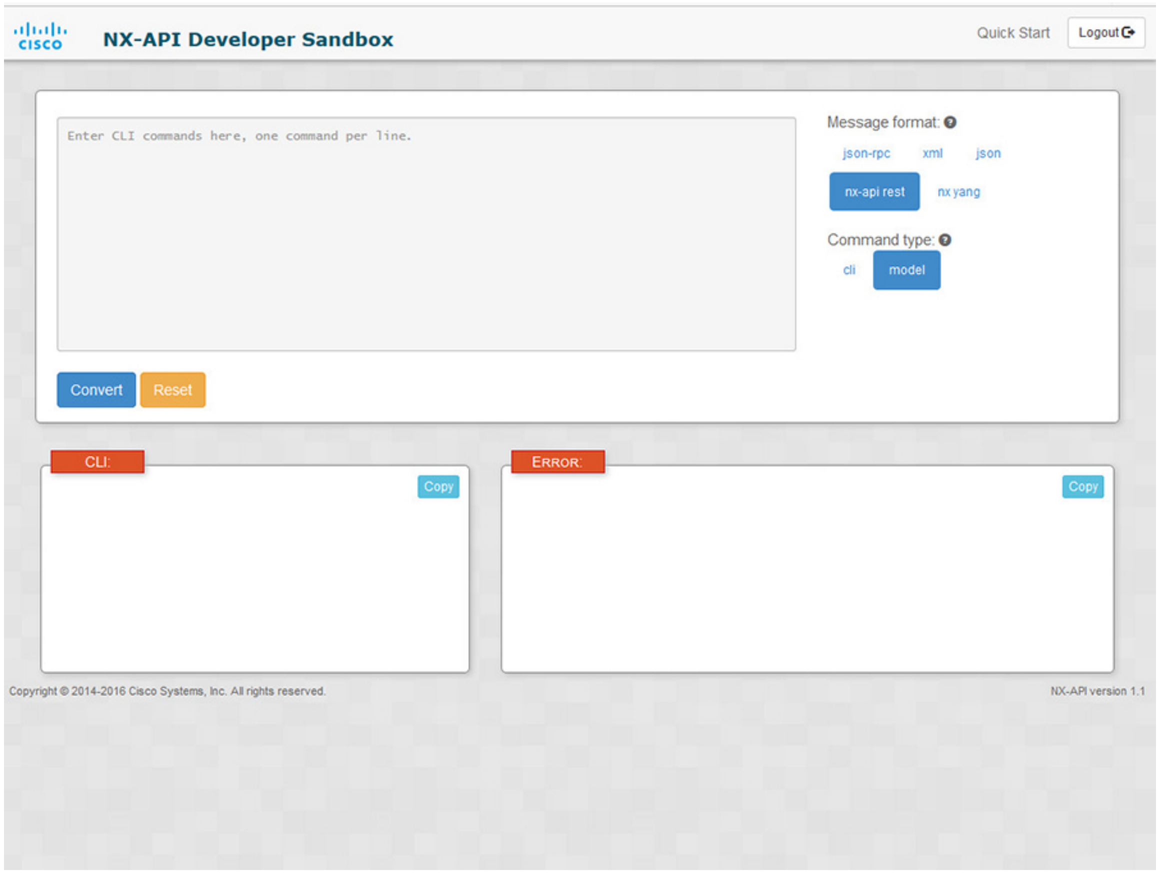
Procedure

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 188.

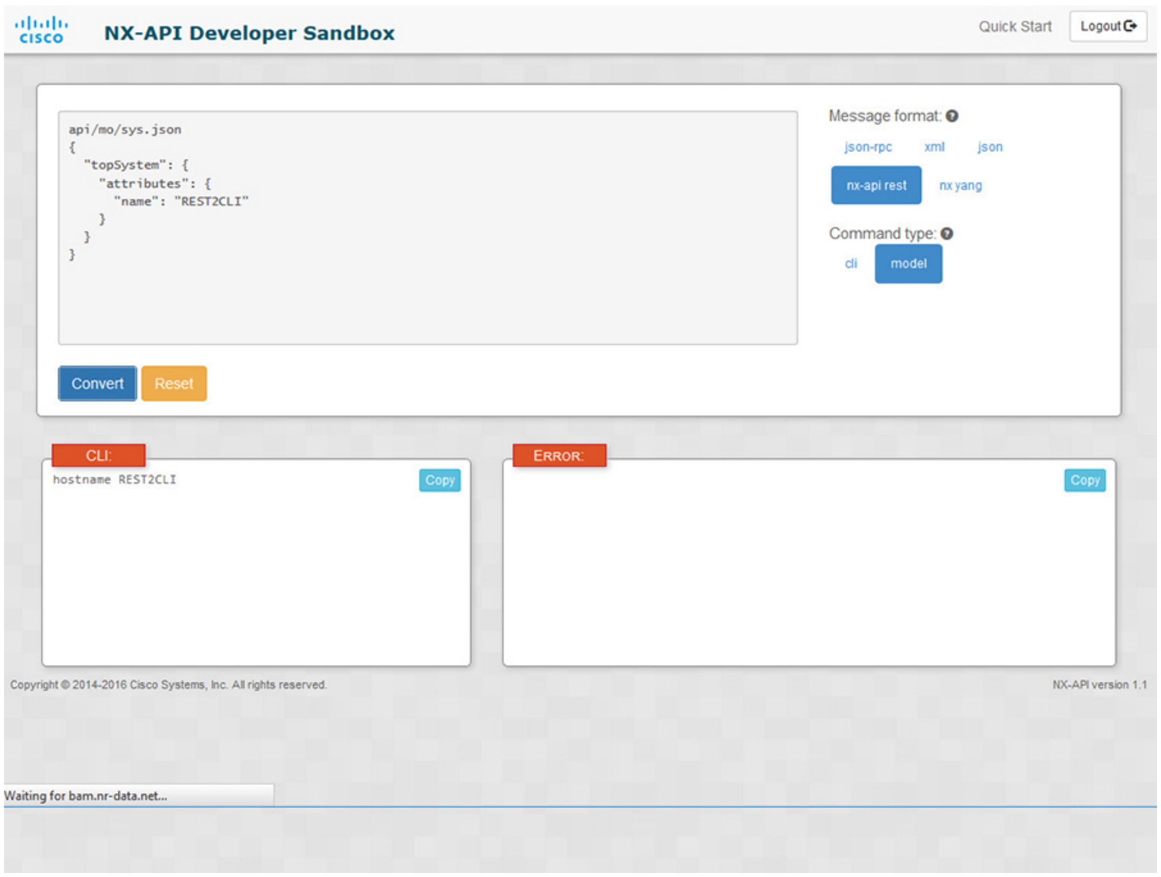
Step 2 Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.

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.



```
api/mo/sys.json
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

```
hostname REST2CLI
```

Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved.

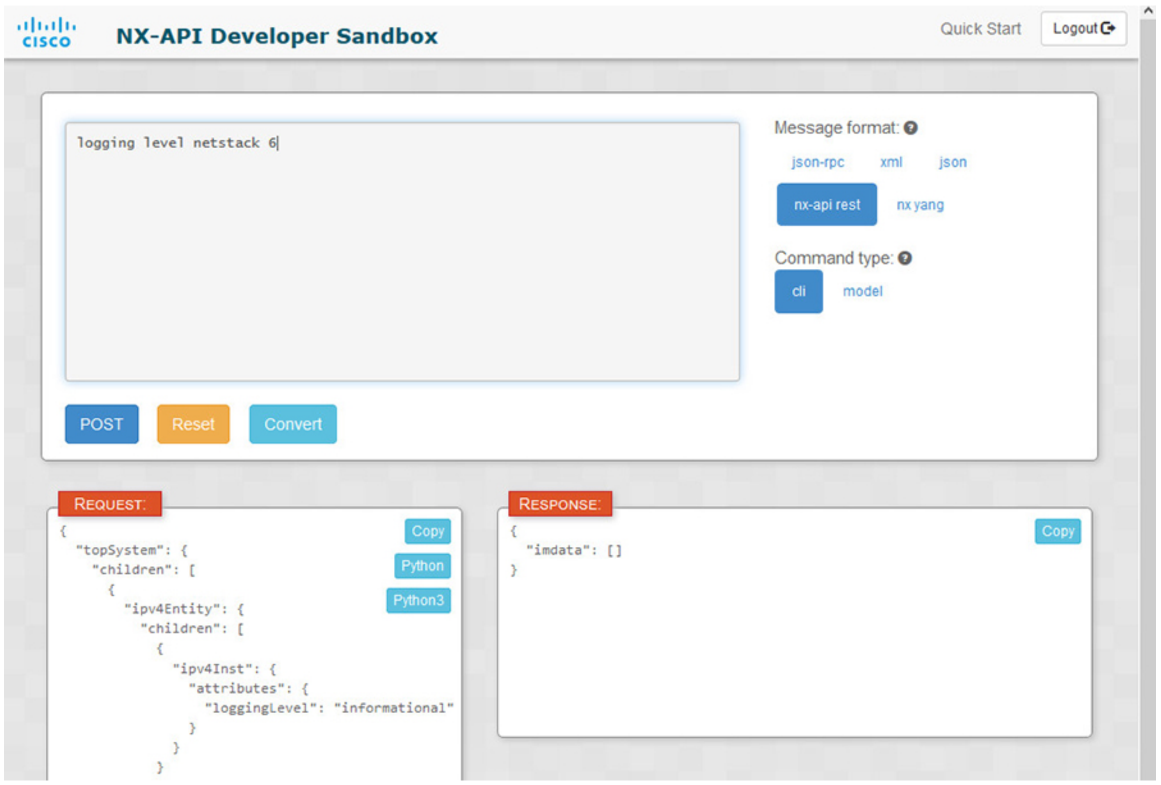
NX-API version 1.1

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.



```
logging level netstack 6
```

```
{
  "topSystem": {
    "children": [
      {
        "ipv4Entity": {
          "children": [
            {
              "ipv4Inst": {
                "attributes": {
                  "loggingLevel": "informational"
                }
              }
            }
```

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

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

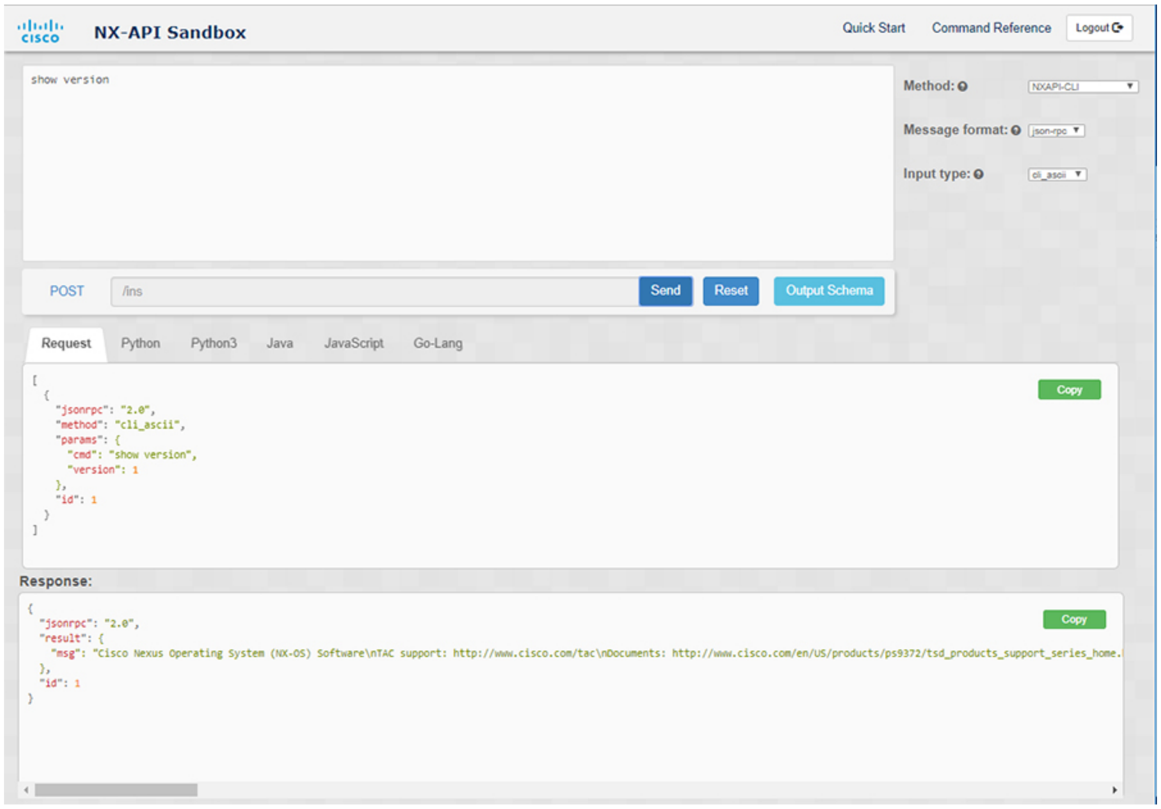
About the NX-API Developer Sandbox

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



```
show version
```

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli_ascii",
    "params": {
      "cmd": "show version",
      "version": 1
    },
    "id": 1
  }
]
```

```
{
  "jsonrpc": "2.0",
  "result": {
    "msg": "Cisco Nexus Operating System (NX-OS) Software\nTAC support: http://www.cisco.com/tac\nDocuments: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.
  },
  "id": 1
}
```

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 200

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 202

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 .

- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- 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 sucessfully.

  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 12: NX-OS API Protocols

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

Protocol	Description
NXAPI-REST (DME)	Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. The NXAPI-REST (DME) protocol displays a drop-down list that enables you to choose from the following methods: • **POST** • **GET** • **PUT** • **DELETE** For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see https://developer.cisco.com/site/cisco-nexus-nx-api-references/.
RESTCONF (Yang)	The YANG ("Yet Another Next Generation") data modeling language for configuration and state data. The RESTCONF (Yang) protocol displays a drop-down list that enables you to choose from the following methods: • **POST** • **GET** • **PUT** • **PATCH** • **DELETE**

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 13: Command Types

Method	Message format	Input/Command type
NXAPI-CLI	json-rpc	• cli — show or configuration commands • cli-ascii — show or configuration commands, output without formatting • 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, [ ].

Method	Message format	Input/Command type
NXAPI-CLI	xml	• cli_show — show commands. If the command does not support XML output, an error message will be returned. • cli_show_ascii — show commands, output without formatting • cli_conf — configuration commands. Interactive configuration commands are not supported. • bash — bash commands. Most non-interactive bash commands are supported. **Note** The bash shell must be enabled in the switch.
NXAPI-CLI	json	• cli_show — show commands. If the command does not support XML output, an error message will be returned. **Note** Beginning with Cisco NX-OS Release 9.3(3), the cli_show_array command is recommended over the cli_show command. • cli_show_array — show commands. Similar to cli_show, but with cli_show_array, data is returned as a list of one element, or an array, within square brackets [ ]. • cli_show_ascii — show commands, output without formatting • cli_conf — configuration commands. Interactive configuration commands are not supported. • bash — bash commands. Most non-interactive bash commands are supported. **Note** The bash shell must be enabled in the switch.
NXAPI-REST (DME)		• cli — CLI to model conversion • model — Model to CLI conversion.
RESTCONF (Yang)	• json — JSON structure is used for payload • xml — XML structure is used for payload	

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 **eoc** (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 14: Chunk Mode Fields

Field Name	Description
Enable Chunk Mode	Click to place a check mark in the **Enable Chunk Mode** 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.
SID	Enter the session Id of the previous response in the **SID** text box to retrieve the next chunk of the response message. **Note** 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

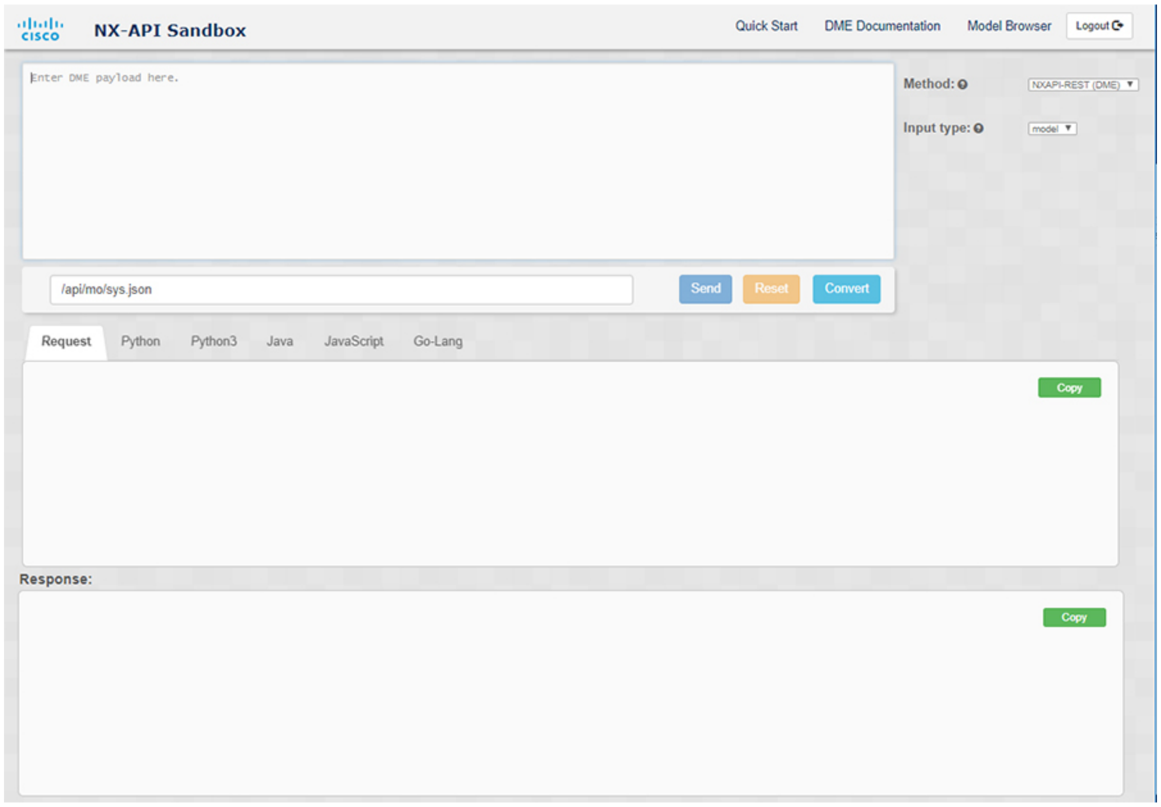
Step 1 Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.

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.



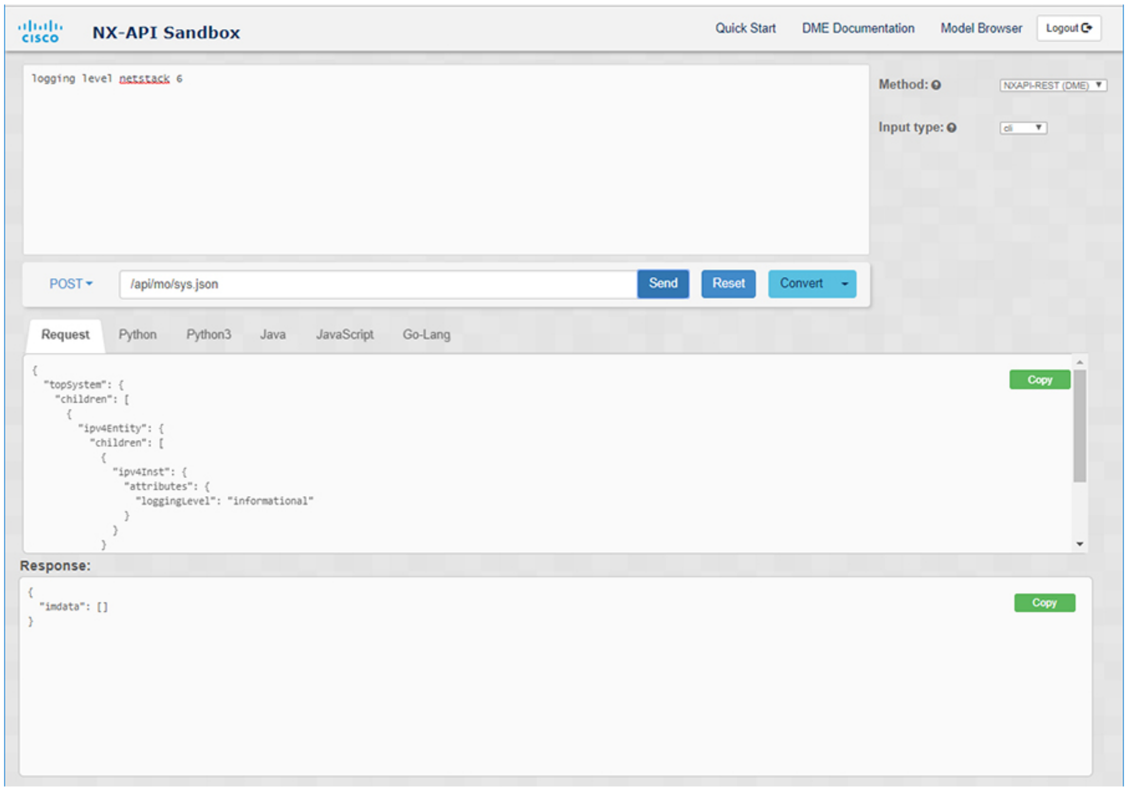
Step 4 Click **Convert**.

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.



```
logging level netstack 6
```

```
/api/mo/sys.json
```

```
{
  "topSystem": {
    "children": [
      {
        "ipv4Entity": {
          "children": [
            {
              "ipv4Inst": {
                "attributes": {
                  "loggingLevel": "informational"
                }
              }
            }
```

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

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 o 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*.

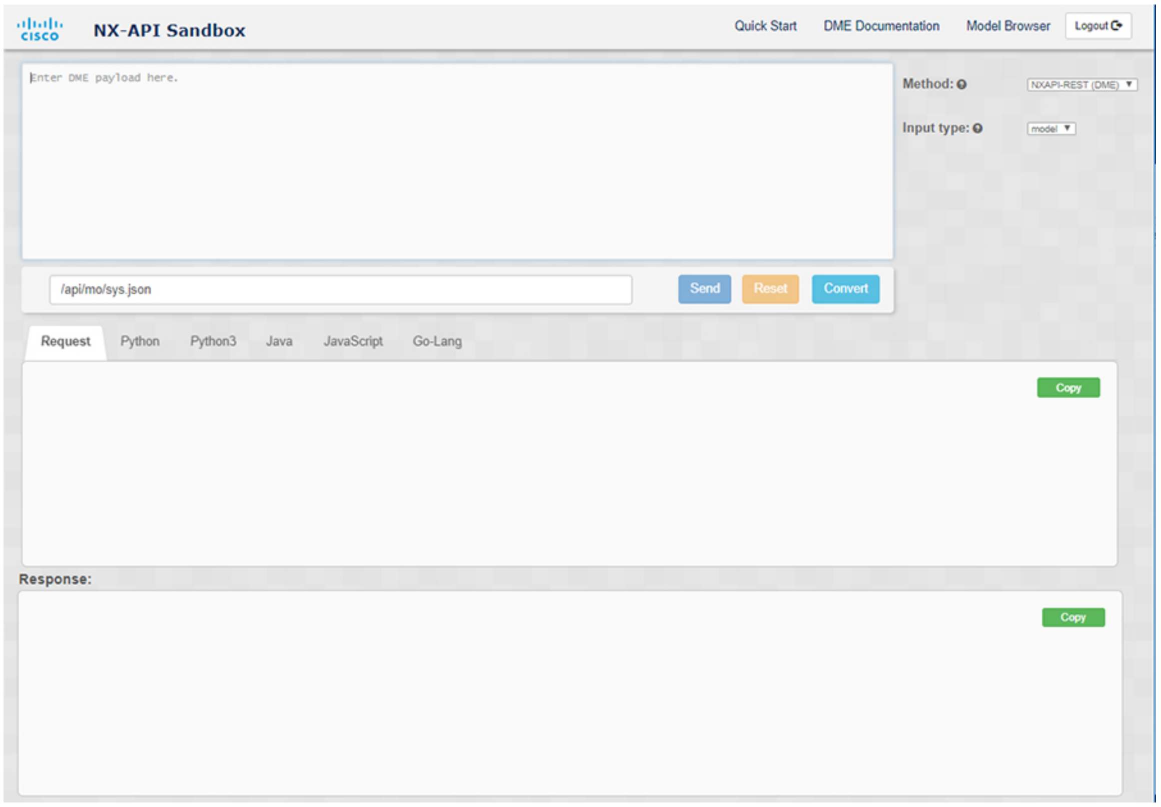
- The top-right corner of the Cisco NX-API Developer Sandbox contains links for additional information. The links that appear depend on the **Method** you choose. The links that appear for the NXAPI-REST (DME) method:
  - **NX-API References**—Enables you to access additional NX-API documentation.
  - **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:



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.

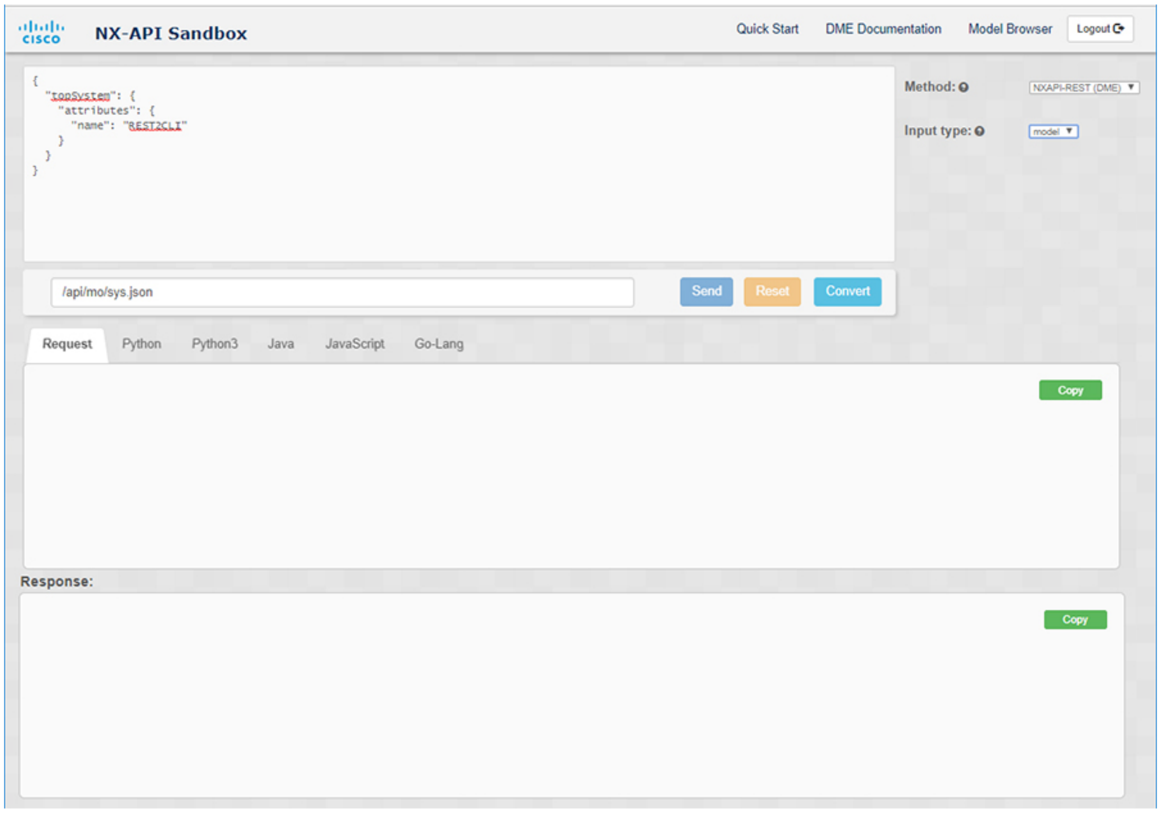
Step 4 Enter the payload in the Command pane.

Step 5 Click **Convert**.

Example:

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

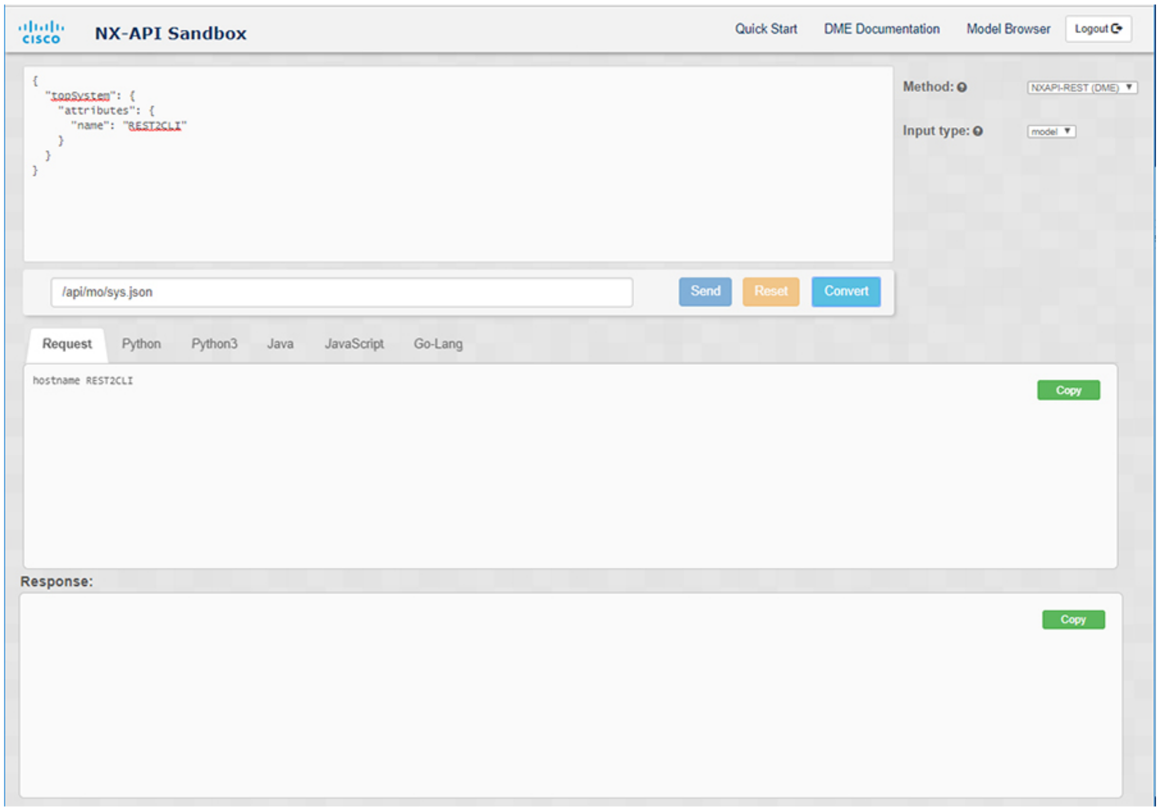
```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



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

```
/api/mo/sys.json
```

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



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

/api/mo/sys.json

```
hostname REST2CLI
```

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 15: Sources of REST2CLI Errors

Payload Issue	Result

The payload contains an attribute that does not exist in the MO.
 	
Example:
 	
`api/mo/sys.json`
`{`
`  "topSystem": {`
`    "children": [`
`      {`
`        "interfaceEntity": {`
`          "children": [`
`            {`
`              "l1PhysIf": {`
`                "attributes": {`
`                  "id": "eth1/1",`
`                  "fakeattribute":`
`"totallyFake"`
`                }`
`              }`
`            }`
`          ]`
`        }`
`      }`
`    ]`
`  }`
`}`
	The **Error** pane will return an error related to the attribute.
	
	Example:
	
	CLI
	
	Error `unknown attribute 'fakeattribute' in element 'l1PhysIf'`

The payload includes MOs that aren't yet supported for conversion:
 	
Example:
 	
`api/mo/sys.json`
`{`
`  "topSystem": {`
`    "children": [`
`      {`
`        "dhcpEntity": {`
`          "children": [`
`            {`
`              "dhcpInst": {`
`                "attributes": {`
`                  "SnoopingEnabled":`
`"yes"`
`                }`
`              }`
`            }`
`          ]`
`        }`
`      }`
`    ]`
`  }`
`}`
	The **Error** Pane will return an error related to the unsupported MO.
	
	Example:
	
	CLI
	
	Error `The entire subtree of "sys/dhcp" is not converted.`

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

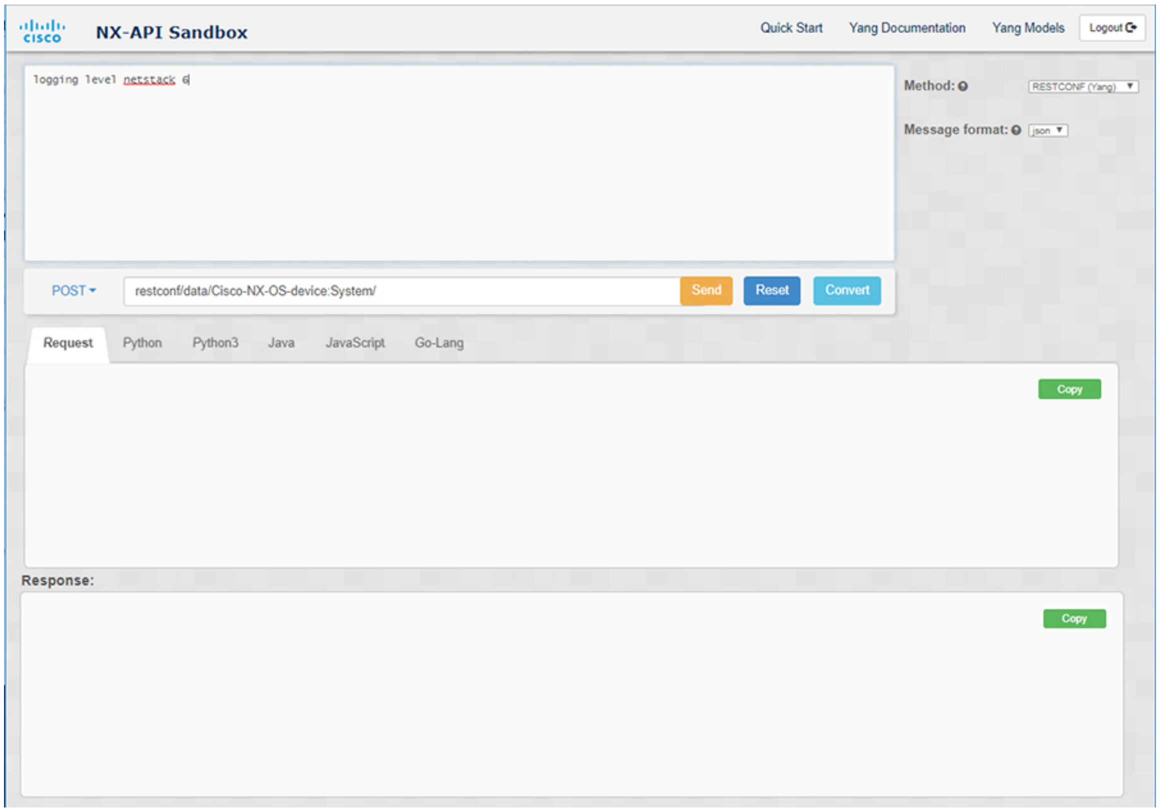
Tip

- 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:



```
logging level netstack 6
```

Method: RESTCONF (Yang)

Message format: json

POST restconf/data/Cisco-NX-OS-device:System/

Step 2 Click **Message format** and choose either **json** or **xml**.

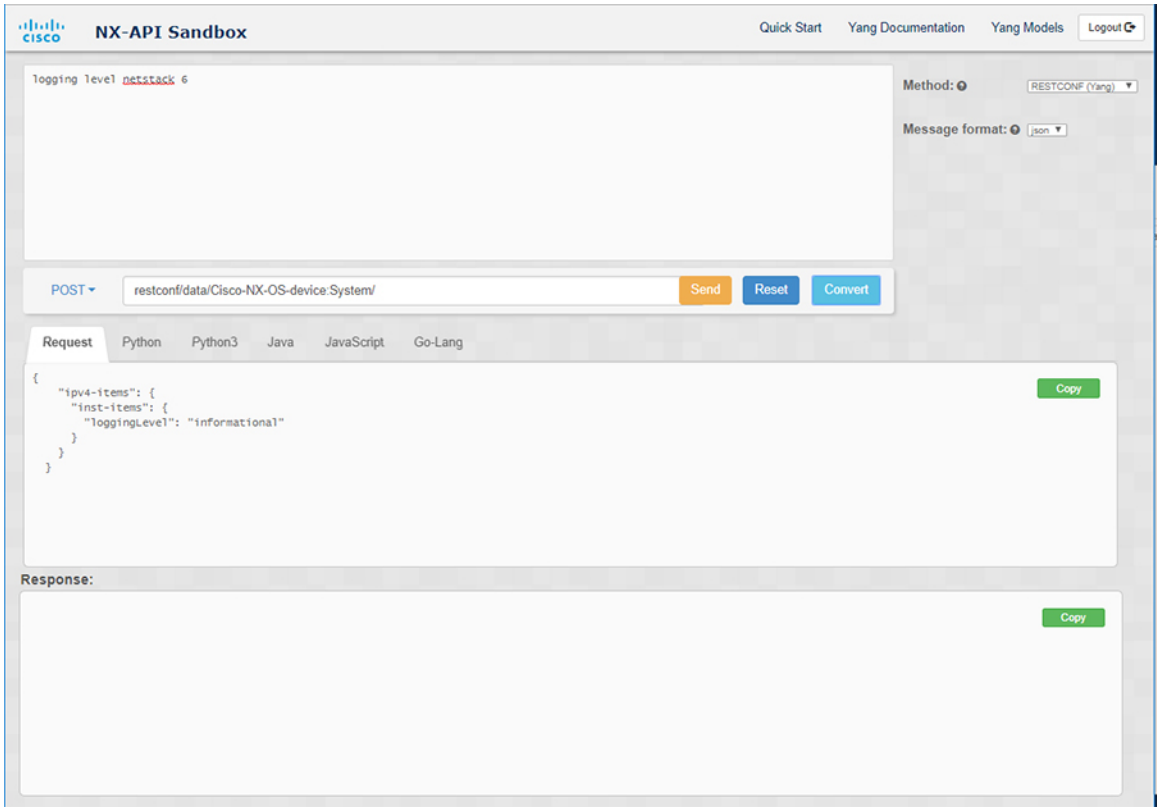
Step 3 Enter a command in the text entry box in the top pane.

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:



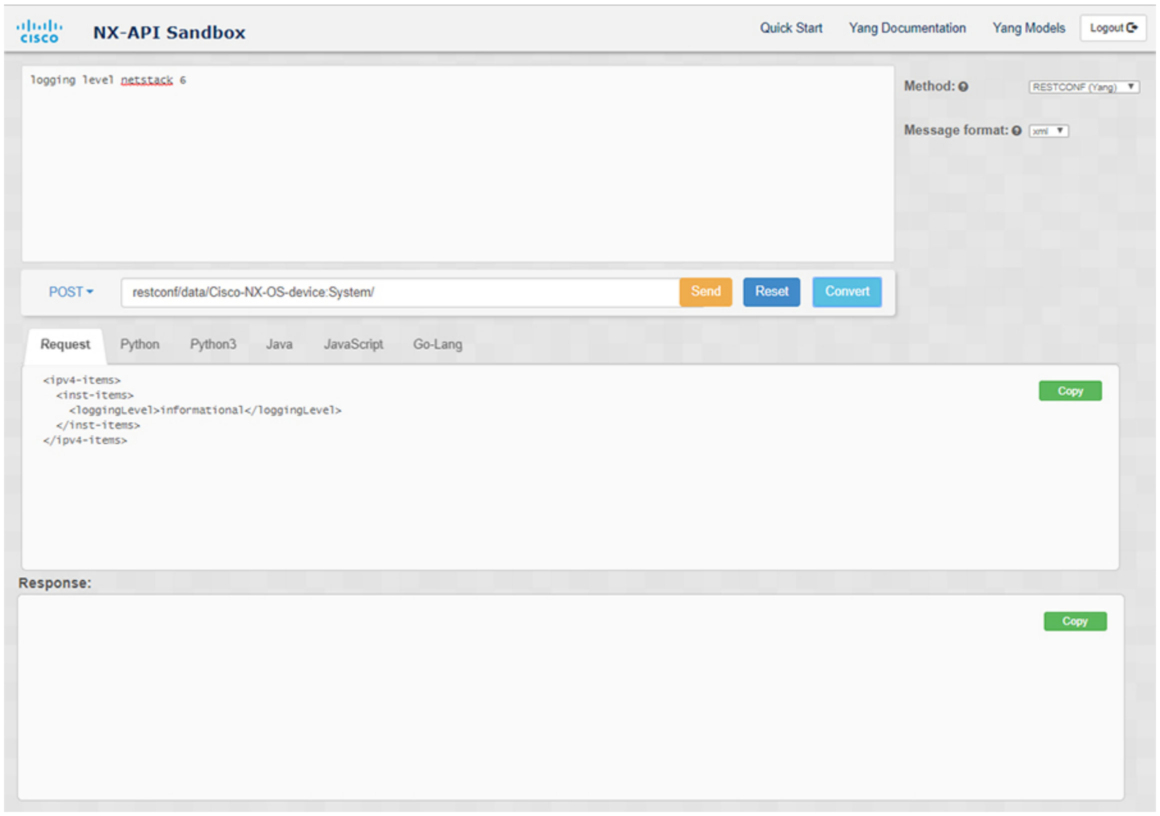
```
logging level netstack 6
```

```
restconf/data/Cisco-NX-OS-device:System/
```

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

Example:

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



```
logging level netstack 6
```

```
restconf/data/Cisco-NX-OS-device:System/
```

```
<ipv4-items>
  <inst-items>
    <loggingLevel>informational</loggingLevel>
  </inst-items>
</ipv4-items>
```

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.

PART IV

Model-Driven Programmability

- Managing Components, on page 213
- Converting CLI Commands to Network Configuration Format, on page 219
- gNMI - gRPC Network Management Interface, on page 225
- Model Driven Telemetry, on page 261

CHAPTER 20

Managing Components

- About the Component RPM Packages, on page 213
- Preparing For Installation, on page 215
- Downloading Components from the Cisco Artifactory, on page 216
- Installing RPM Packages, on page 217

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 unistall 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
```

Downloading Components from the Cisco Artifactory

The NX-OS Programmable Interface Component RPMs can be downloaded from the Cisco Artifactory at the following URL. The RPMs are organized by NX-OS release-specific directories. Ensure that you are downloading the RPMs from the correct NX-OS release directory.

https://devhub.cisco.com/artifactory/open-nxos-agents

The NX-OS Programmable Interface Component RPMs adhere to the following naming convention:

*<package>-<version>-<NX-OS release>.<architecture>***.rpm**

Select and download the desired NX-OS Programmable Interface Component RPM packages to the device for installation as described in the following sections.

Installing RPM Packages

Installing the Programmable Interface Base And Common Model Component RPM Packages

Before you begin

- From the Cisco Artifactory, download the following packages:
    - mtx-infra
    - mtx-device
    - mtx-netconf-agent/mtx-restconf-agent/mtx-grpc-agent (at least one)
    - mtx-openconfig-all (alternatively, selected individual models)
- Using the CLI commands in Verify Device Readiness, on page 216, confirm that all line cards in the Active and Standby devices are up and ready.

Procedure

Step 1 Copy the downloaded RPMs to the device.

Example:

```
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
 vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
 bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
 bootflash: vrf management
```

Step 2 From the Bash shell, install the RPMs.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

Step 3 From the Bash shell, verify the installation.

Example:

```
bash-4.2# yum list installed | grep mtx
```

CHAPTER 21

Converting CLI Commands to Network Configuration Format

- Information About XMLIN, on page 219
- Licensing Requirements for XMLIN, on page 219
- Installing and Using the XMLIN Tool, on page 220
- Converting Show Command Output to XML, on page 220
- Configuration Examples for XMLIN, on page 221

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 16: 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 *Cisco NX-OS Licensing Guide*.

Installing and Using the XMLIN Tool

You can install the XMLIN tool and then use it to convert configuration commands to NETCONF format.

Before you begin

The XMLIN tool can generate NETCONF instances of commands even if the corresponding feature sets or required hardware capabilities are not available on the device. But, you might still need to install some feature sets before entering the **xmlin** command.

Procedure

	Command or Action	Purpose
Step 1	switch# **xmlin**	
Step 2	switch(xmlin)# **configure terminal**	Enters global configuration mode.
Step 3	Configuration commands	Converts configuration commands to NETCONF format.
Step 4	(Optional) switch(config)(xmlin)# **end**	Generates the corresponding <edit-config> request. **Note** Enter the **end** command to finish the current XML configuration before you generate an XML instance for a **show** command.
Step 5	(Optional) switch(config-if-verify)(xmlin)# **show** *commands*	Converts **show** commands to NETCONF format.
Step 6	(Optional) switch(config-if-verify)(xmlin)# **exit**	Returns to EXEC mode.

Converting Show Command Output to XML

You can convert the output of show commands to XML.

Before you begin

Make sure that all features for the commands you want to convert are installed and enabled on the device. Otherwise, the commands fail.

You can use the **terminal verify-only** command to verify that a feature is enabled without entering it on the device.

Make sure that all required hardware for the commands you want to convert are present on the device. Otherwise, the commands fail.

Make sure that the XMLIN tool is installed.

Procedure

	Command or Action	Purpose
Step 1	switch# *show-command* | **xmlin**	Enters global configuration mode. **Note** You cannot use this command with configuration commands.

Configuration Examples for XMLIN

The following example shows how the XMLIN tool is installed on the device and used to convert a set of configuration commands to an <edit-config> instance.

```
switch# xmlin
******************************************
Loading the xmlin tool. Please be patient.
******************************************
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright ©) 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:m1="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>
              <m1:cdp>
                <m1:enable/>
              </m1: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 22

gNMI - gRPC Network Management Interface

This chapter contains the following topics:

- About gNMI, on page 225
- gNMI RPC and SUBSCRIBE, on page 226
- Guidelines and Limitations for gNMI, on page 227
- Configuring gNMI, on page 229
- Configuring Server Certificate, on page 230
- Generating Key/Certificate Examples , on page 231
- Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(2) and Earlier, on page 231
- Examples for Generating and Configuring Key/Certificate for Cisco NX-OS Release 9.3(3) and Later, on page 232
- Verifying gNMI, on page 234
- Clients, on page 240
- Sample DME Subscription - PROTO Encoding, on page 240
- Capabilities, on page 241
- Get, on page 245
- Set, on page 246
- Subscribe, on page 247
- Streaming Syslog, on page 251
- Troubleshooting, on page 257

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.

Table 17: 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 18: 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.
	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.
- 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.

  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.

- 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 gNMI feature supports the Subscribe and Capability gNMI RPCs.
- 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 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 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.

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.

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.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enters global configuration mode.

Step 2	**feature grpc**
	Example:
	`switch-1# feature grpc`
	`switch-1(config)#`
		Enables the gRPC agent, which supports the gNMI interface for dial-in.

Step 3	(Optional) **grpc port** *port-id* **Example:** `switch-1(config)# grpc port 50051`	Configure the port number. The range of *port-id* is 1024–65535. 50051 id the default. **Note** This command is available beginning with Cisco NX-OS Release 9.3(3).
Step 4	(Optional) **grpc certificate** *certificate-id* **Example:** `switch-1(config)# grpc certificate cert-1`	Specify the certificate trustpoint ID. For more information, see the Installing Identity Certificates section of the Cisco Nexus Series NX-OS Security Configuration Guide, Release 9.3(x) for importing the certificate to the switch. **Note** This command is available beginning with Cisco NX-OS Release 9.3(3).

	Command or Action	Purpose

Step 5	**grpc gnmi max-concurrent-call** *number*
	Example:
	`switch-1(config)# grpc gnmi max-concurrent-call 16`
	`switch-1(config)#`
		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.
		Note The configured limit does not affect the gRPCConfigOper service.

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.

```
#show grpc gnmi service statistics

=============
gRPC Endpoint
=============

Vrf            : management
Server address : [::]:50051

Cert notBefore : Mon Jan 27 15:34:08 PDT 2020
Cert notAfter  : Tue Jan 26 15:34:08 PDT 2021

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
```

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.

Note After the certificate expires, there are two ways to have the key/certificate to regenerate:

- Reload the switch.
- Manually delete the key/certificate in the `/opt/mtx/etc` folder and enter the **no feature grpc** and **feature grpc** commands.

Generating Key/Certificate Examples

Follow these examples to generate Key/Certificates:

- Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(2) and Earlier, on page 231

Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(2) and Earlier

The following is an example for generating key/certificate:

For more information on generating identify 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
```

Step 2 After generating the key and pem files, modify the mtx.conf.user files in the Bash shell to have the gRPC service pick up the certificates.

```
[grpc]
key = /bootflash/self-sign2048.key
cert = /bootflash/self-sign2048.pem
```

Step 3 Reload the box to have the gRPC service pick up the certificate.

Step 4 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
```

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
```

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 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 4 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 5 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
```

Verifying gNMI

To verify the gNMI configuration, enter the following command:

Command	Description
show grpc gnmi service statistics	Displays a summary of the agent running status, respectively for the management VRF, or the default VRF (if configured). It also displays: • Basic overall counters • Certificate expiration time **Note** If the certificate is expired, the agent cannot accept requests.

Command	Description
show grpc gnmi rpc summary	Displays the following: • Number of capability RPCs received. • Capability RPC errors. • Number of Get RPCs received. • Get RPC errors. • Number of Set RPCs received. • Set RPC errors. • More error types and counts.

Command	Description
show grpc gnmi transactions	The **show grpc gnmi transactions** 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: • RPC – This shows the type of RPC that was received (Get, Set, Capabilities) • DataType – For a Get only. Has values ALL, CONFIG, and STATE. • 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. • Time In -- shows timestamp of when the RPC was received by the gNMI handler. • Duration – time delta in ms from receiving the request to giving response. • Status – the status code of the operation returned to the client `(0 = Success, !0 == error).` This section is data that is kept per path within a single gNMI transaction. For example, a single Get or Set • subtype – for a Set RPC, shows the specific operation that is requested per path (Delete, Update, Replace). For Get, there is no subtype. • dtx – shows that this path was processed in DTX "fast" path or not. A dash '-' means no, an asterisk '*' means yes. • st – Status for this path. The meaning is as follows: • OK: path is valid and processed by infra successfully. • ERR: path is either invalid or generated error by infra • --: path not processed yet, might or might not be valid and has not been sent to infra yet. • path – the path

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][n
hVrf=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
```

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.

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 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

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

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_bl/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 name="format">
```

```
        <leaf name="content" type="string" default="$DATETIME$
$COMPONENTID$ $TYPE$: $MSG$">$DATETIME$ $COMPONENTID$ $TYPE$
$SRCFILE$ @ $SRCLINE$ $FCNINFO$:$MSG$</leaf>
            <container name="componentID">
          <leaf name="enabled" type="boolean" default="true"></leaf>
            </container>
            <container name="dateTime">
          <leaf name="enabled" type="boolean" default="true"></leaf>
          <leaf name="format" type="string" default="%y%m%d.%H%M%S"><
/leaf>
             </container>
             <container name="fcn">
           <leaf name="enabled" type="boolean" default="true"></leaf>
           <leaf name="format" type="string"
default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></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 /volaiflels -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 23

Model Driven Telemetry

- About Telemetry, on page 261
- Licensing Requirements for Telemetry, on page 263
- Installing and Upgrading Telemetry, on page 263
- Guidelines and Limitations for Model Driven Telemetry, on page 264
- Configuring Telemetry Using the CLI, on page 269
- Configuring Telemetry Using the NX-API, on page 288
- Telemetry Path Labels, on page 302
- Native Data Source Paths, on page 316
- Streaming Syslog, on page 327
- Additional References, on page 333

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.
- **Supervisor Failover** — Although telemetry is not on hot standby, telemetry configuration and streaming services are restored when the new active supervisor is running.
- **Process Restart** — If the telemetry process freezes or restarts for any reason, configuration, and streaming services are restored when telemetry 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 *Cisco NX-OS Licensing Guide*.

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_n3000.rpm**

As in the following example:

```
    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:

```
yum 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:

```
    yum downgrade telemetry
```

Verifying the Active Version

To verify the active version, run the following command from the switch `exec` prompt:

```
show install active
```

Note The `show install active` command 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.

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"
                      }
                    }
                  }
                ]
              }
            }
```

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"
                      }
                    }
                  }
                ]
              }
            }
```

Support for Node ID

Beginning in NX-OS release 9.3.1, you can configure a custom Node ID string for a telemetry receiver through the **use-nodeid** command. By default, the hostname 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
```

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.

Configuring Telemetry Using the CLI

Configuring Telemetry Using the NX-OS CLI

The following steps enable streaming telemetry and configuring the source and destination of the data stream. These steps also include optional steps to enable and configure SSL/TLS certificates and GPB encoding.

Procedure

	Command or Action	Purpose

Step 1	(Optional) **openssl** *argument*
	Example:
	Generate an SSL/TLS certificate using a specific argument, such as the following:
	• To generate a private RSA key: **openssl genrsa** *-cipher* **-out** *filename.key cipher-bit-length*
	For example:
	`switch# openssl genrsa -des3 -out server.key 2048`
	• To write the RSA key: **openssl rsa -in** *filename.key* **-out** *filename.key*
	For example:
	`switch# openssl rsa -in server.key -out server.key`
	• To create a certificate that contains the public or private key: **openssl req**
	-encoding-standard **-new -new** *filename.key* **-out** *filename.csr* **-subj '/CN=localhost'**
	For example:
	`switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'`
	• To create a public key: **openssl x509 -req** *-encoding-standard* **-days** *timeframe* **-in** *filename.csr* **-signkey** *filename.key* **-out** *filename.csr*
		Create an SSL or TLS certificate on the server that receives the data, where the `private.key` file is the private key and the `public.crt` is the public key.

	Command or Action	Purpose
	For example: `switch# openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt`	

Step 2	**configure terminal**
	Example:
	`switch# configure terminal`
	`switch(config)#`
		Enter the global configuration mode.

Step 3	**feature telemetry**	Enable the streaming telemetry feature.
Step 4	**feature nxapi**	Enable NX-API.
Step 5	**nxapi use-vrf management**	Enable the VRF management to be used for NX-API communication.

Step 6	**telemetry**
	Example:
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		Enter configuration mode for streaming telemetry.

Step 7	(Optional) **certificate** *certificate_path host_URL* **Example:** `switch(config-telemetry)# certificate /bootflash/server.key localhost`	Use an existing SSL/TLS certificate.

Step 8	(Optional) Specify a transport VRF or enable telemetry compression for gRPC transport.
	Example:
	`switch(config-telemetry)# destination-profile`
	`switch(conf-tm-dest-profile)# use-vrf default`
	`switch(conf-tm-dest-profile)# use-compression gzip`
	`switch(conf-tm-dest-profile)# use-retry size 10`
	`switch(conf-tm-dest-profile)# source-interface loopback1`
		• Enter the **destination-profile** command to specify the default destination profile.
		• Enter any of the following commands:
		  • **use-vrf** *vrf* to specify the destination VRF.
		  • **use-compression gzip** to specify the destination compression method.
		  • **use-retry size** *size* to specify the send retry details, with a retry buffer size between 10–1500 megabytes.
		  • **source-interface** *interface-name* to stream data from the configured interface to a destination with the source IP address.

	Command or Action	Purpose
		Note After configuring the **use-vrf** command, you must configure a new destination IP address within the new VRF. However, you may re-use the same destination IP address by unconfiguring and reconfiguring the destination. This action ensures that the telemetry data streams to the same destination IP address in the new VRF.

Step 9	**sensor-group** *sgrp_id*
	Example:
	`switch(config-telemetry)#` **`sensor-group 100`**
	`switch(conf-tm-sensor)#`
		Create a sensor group with ID *srgp_id* and enter sensor group configuration mode.
		Currently only numeric ID values are supported. The sensor group defines nodes that will be monitored for telemetry reporting.

Step 10	(Optional) **data-source** *data-source-type* **Example:** `switch(config-telemetry)#` **`data-source NX-API`**	Select a data source. Select from either YANG, DME or NX-API as the data source. **Note** DME is the default data source.
Step 11	**path** *sensor_path* **depth 0** [**filter-condition** *filter*] [**alias** *path_alias*] **Example:** • The following command is applicable for DME, not for NX-API or YANG: `switch(conf-tm-sensor)#` **`path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")`** Use the following syntax for state-based filtering to trigger only when **`operSt`** changes from **`up`** to **`down`**, with no notifications of when the MO changes. `switch(conf-tm-sensor)#` **`path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))`** Use the following syntax to distinguish the path on the UTR side. `switch(conf-tm-sensor)#` **`path sys/ch/ftslot-1/ft alias ft_1`** • The following command is applicable for NX-API, not for DME or YANG:	Add a sensor path to the sensor group. • Beginning with the Cisco NX-OS 9.3(5) release, the **alias** keyword is introduced. • The **depth** setting specifies the retrieval level for the sensor path. Depth settings of **0** - **32**, **unbounded** are supported. **Note** **depth 0** is the default depth. NX-API-based sensor paths can only use **depth 0**. If a path is subscribed for the event collection, the depth only supports 0 and unbounded. Other values would be treated as 0. • The optional **filter-condition** parameter can be specified to create a specific filter for event-based subscriptions. For state-based filtering, the filter returns both when a state has changed and when an event has occurred during the specified

	Command or Action	Purpose

	`switch(conf-tm-sensor)# ` **`path "show interface" depth 0`**
	
	• The following command is applicable for device YANG:
	
	`switch(conf-tm-sensor)# ` **`path Cisco-NX-OS-device:System/bgp-items/inst-items`**
	
	• The following commands are applicable for OpenConfig YANG:
	
	`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`**
	
	• The following command is applicable for NX-API:
	
	`switch(conf-tm-sensor)# ` **`path "show interface" depth 0 alias sh_int_alias`**
	
	• The following command is applicable for OpenConfig:
	
	`switch(conf-tm-sensor)# ` **`path openconfig-bgp:bgp alias oc_bgp_alias`**
		state. That is, a filter condition for the DN **sys/bd/bd-[vlan]** of **eq(l2Bd.operSt, "down")** triggers when the operSt changes, and when the DN's property changes while the operSt remains **down**, such as a **no shutdown** command is issued while the VLAN is operationally **down**.
		
		Note query-condition parameter — For DME, based on the DN, the query-condition parameter can be specified to fetch MOTL and ephemeral data with the following syntax: query-condition "rsp-foreign-subtree=applied-config"; query-condition "rsp-foreign-subtree=ephemeral".
		
		• For the YANG model, the sensor path format is as follows: *module_name*: *YANG_path*, where *module_name* is the name of the YANG model file. For example:
		
		• For device YANG:
		
		Cisco-NX-OS-device:System/bgp-items/inst-items
		
		• For OpenConfig YANG:
		
		openconfig-bgp:bgp
		
		Note The **depth**, **filter-condition**, and **query-condition** parameters are not supported for YANG currently.
		
		For the openconfig YANG models, go to https://github.com/YangModels/yang/tree/master/vendor/cisco/nx and navigate to the appropriate folder for the latest release.
		
		Instead of installing a specific model, you can install the openconfig-all RPM which has all the OpenConfig models.
		
		For example:

	Command or Action	Purpose
		install add mtx-openconfig-bgp-1.0.0.0-7.0.3.IHD8.1.lib32_n9000.rpm activate

Step 12	**destination-group** *dgrp_id*
	Example:
	`switch(conf-tm-sensor)#`
	`destination-group 100`
	`switch(conf-tm-dest)#`
		Create a destination group and enter destination group configuration mode.
		Currently *dgrp_id* only supports numeric ID values.

Step 13	(Optional) **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)#` **`ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`**
	`switch(conf-tm-sensor)#` **`ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON`**
	`switch(conf-tm-sensor)#` **`ip address 171.70.55.69 port 50009 protocol UDP encoding JSON`**
		Specify an IPv4 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 14	(Optional) **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)#` **`ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`**
	`switch(conf-tm-sensor)#` **`ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON`**
	`switch(conf-tm-sensor)#` **`ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON`**
		Specify an IPv6 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 15	*ip_version* **address** *ip_address* **port** *portnum* **Example:** • For IPv4: `switch(conf-tm-dest)#` **`ip address 1.2.3.4 port 50003`** • For IPv6: `switch(conf-tm-dest)#` **`ipv6 address 10:10::1 port 8000`**	Create a destination profile for the outgoing data, where *ip_version* is either ip (for IPv4) or ipv6 (for IPv6). 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.

	Command or Action	Purpose
Step 16	(Optional) **use-chunking size** *chunking_size* **Example:** `switch(conf-tm-dest)# use-chunking size 64`	Enable gRPC chunking and set the chunking size, between 64-4096 bytes. See the section "Support for gRPC Chunking" for more information.

Step 17	**subscription** *sub_id*
	Example:
	`switch(conf-tm-dest)# subscription 100`
	`switch(conf-tm-sub)#`
		Create a subscription node with ID and enter the subscription configuration mode.
		Currently *sub_id* only supports numeric ID values.
		Note When subscribing to a DN, check whether the DN is supported by DME using REST to ensure that events will stream.

Step 18	**snsr-grp** *sgrp_id* **sample-interval** *interval* **Example:** `switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000`	Link the sensor group with ID *sgrp_id* 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.
Step 19	**dst-grp** *dgrp_id* **Example:** `switch(conf-tm-sub)# dst-grp 100`	Link the destination group with ID *dgrp_id* 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.

Procedure

	Command or Action	Purpose

Step 1	**show telemetry control database sensor-groups**
	
	Example:
	```
	switch-1# show telemetry control database
	 sensor-groups
	Sensor Group Database size = 2
	----------------------------------------------------
	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
	Streaming Time in ms (Cur/Min/Max):
	2515/2356/28403

	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
	Streaming Time in ms (Cur/Min/Max):
	149/147/23548

	Collection Statistics:
	  collection_id_dropped      = 0
	  last_collection_id_dropped = 0
	  drop_count                 = 0

	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
	  subscription 1
	```
		Calculate the total streaming time.
		
		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).
		
		Compare the configured cadence to the total streaming time for the sensor group.
		
		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).

	Command or Action	Purpose

	`dst-grp 1`
	`snsr-grp 1 sample-interval 5000`
	`snsr-grp 2 sample-interval 5000`	

Step 2	**sensor group** *number* **Example:** `switch-1(config-telemetry)#` **`sensor group1`**	If the total streaming time is not less than the cadence, enter the sensor group for which you want to set the interval.
Step 3	**subscription** *number* **Example:** `switch-1(conf-tm-sensor)#` **`subscription 100`**	Edit the subscription for the sensor group.
Step 4	**snsr-grp** *number* **sample-interval** *milliseconds* **Example:** `switch-1(conf-tm-sub)#` **`snsr-grp number sample-interval 5000`**	For the appropriate sensor group, set the sample interval to a value greater than the total streaming time. 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.

Step 5	**show system resources**
	Example:
	`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`
		Check the CPU usage.
		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.

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
```

Displaying Telemetry Log and Trace Information

Use the following NX-OS CLI commands to display the log and trace information.

show tech-support telemetry

This NX-OS CLI command collects the telemetry log contents from the tech-support log. In this example, the command output is redirected into a file in bootflash.

```
switch# show tech-support telemetry > bootflash:tmst.log
```

show system internal telemetry trace

The **show system internal telemetry trace** [**tm-events** | **tm-errors** |**tm-logs** | **all**] command displays system internal telemetry trace information.

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy dest
 profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
1 does not have any sensor groups

3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#
switch# show system internal telemetry trace tm-events
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#
```

Configuring Telemetry Using the NX-API

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in 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 configured to run the NX-API from the CLI:

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

```
nxapi use-vrf vrf_name
nxapi http port port_number
```

Procedure

	Command or Action	Purpose

Step 1	Enable the telemetry feature.
	
	Example:
	```
	{
	  "fmEntity" : {
	    "children" : [{
	      "fmTelemetry" : {
	        "attributes" : {
	          "adminSt" : "enabled"
	        }
	      }
	    }
	    ]
	  }
	}
	```
		The root element is **fmTelemetry** and the base path for this element is `sys/fm`. Configure the **adminSt** attribute as `enabled`.

Step 2	Create the root level of the JSON payload to describe the telemetry configuration.
	
	Example:
	```
	{
	    "telemetryEntity": {
	        "attributes": {
	            "dn": "sys/tm"
	        },
	    }
	}
	```
		The root element is **telemetryEntity** and the base path for this element is `sys/tm`. Configure the **dn** attribute as `sys/tm`.

Step 3	Create a sensor group to contain the defined sensor paths.
	
	Example:
	```
	"telemetrySensorGroup": {
	    "attributes": {
	        "id": "10",
	        "rn": "sensor-10"
	        "dataSrc": "NX-API"
	    },  "children": [{
	    }]
	}
	```
		A telemetry sensor group is defined in an object of class **telemetrySensorGroup**. Configure the following attributes of the object:
		
		• **id** — An identifier for the sensor group. Currently only numeric ID values are supported.
		
		• **rn** — The relative name of the sensor group object in the format: **sensor-***id*.
		
		• **dataSrc** — Selects the data source from **DEFAULT**, **DME**, **YANG**, or **NX-API**.
		
		Children of the sensor group object include sensor paths and one or more relation objects

	Command or Action	Purpose
		(**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.

Step 4	(Optional) Add an SSL/TLS certificate and a host.
	
	Example:
	```
	{
	    "telemetryCertificate": {
	        "attributes": {
	            "filename": "root.pem"
	             "hostname": "c.com"
	         }
	    }
	}
	```
		The **telemetryCertificate** defines the location of the SSL/TLS certificate with the telemetry subscription/destination.

Step 5	Define a telemetry destination group.
	
	Example:
	```
	{
	      "telemetryDestGroup": {
	        "attributes": {
	          "id": "20"
	        }
	      }
	}
	```
		A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.

Step 6	Define a telemetry destination profile.
	
	Example:
	```
	{
	   "telemetryDestProfile": {
	      "attributes": {
	         "adminSt": "enabled"
	      },
	      "children": [
	         {

	"telemetryDestOptSourceInterface": {
	               "attributes": {
	                  "name": "lo0"
	               }
	            }
	         }
	      ]
	   }
	}
	```
		A telemetry destination profile is defined in **telemetryDestProfile**.
		
		• Configure the **adminSt** attribute as `enabled`.
		
		• Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream data from the configured interface to a destination with the source IP address.

Step 7	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
	
	Example:
	```
	{
	    "telemetryDest": {
	        "attributes": {
	```
		A telemetry destination is defined in an object of class **telemetryDest**. Configure the following attributes of the object:
		
		• **addr** — The IP address of the destination.
		
		• **port** — The port number of the destination.

	Command or Action	Purpose

	```
	"addr": "1.2.3.4",
	"enc": "GPB",
	"port": "50001",
	"proto": "gRPC",
	"rn":
	"addr-[1.2.3.4]-port-50001"
	}
	}
	}
	```
		• **rn** — The relative name of the destination object in the format: **path-[***path***]**.
		• **enc** — The encoding type of the telemetry data to be sent. NX-OS supports:
		  • Google protocol buffers (GPB) for gRPC.
		  • JSON for C.
		  • GPB or JSON for UDP and secure UDP (DTLS).
		• **proto** — The transport protocol type of the telemetry data to be sent. NX-OS supports:
		  • gRPC
		  • HTTP
		  • VUDP and secure UDP (DTLS)
		• Supported encoded types are:
		  • HTTP/JSON YES
		  • HTTP/Form-data YES Only supported for Bin Logging.
		  • GRPC/GPB-Compact YES Native Data Source Only.
		  • GRPC/GPB YES
		  • UDP/GPB YES
		  • UDP/JSON YES

Step 8	Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.
	Example:
	```
	{
	    "telemetryDestGrpOptChunking": {
	        "attributes": {
	            "chunkSize": "2048",
	            "dn":
	"sys/tm/dest-1/chunking"
	        }
	    }
	}
	```
		See #unique_289 unique_289_Connect_42_section_ebt_1kr_h2b for more information.

	Command or Action	Purpose

Step 9	Create a telemetry subscription to configure the telemetry behavior.
	
	Example:
	```
	"telemetrySubscription": {
	    "attributes": {
	        "id": "30",
	        "rn": "subs-30"
	    },  "children": [{
	    }]
	}
	```
		A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object:
		
		• **id** — An identifier for the subscription. Currently only numeric ID values are supported.
		
		• **rn** — The relative name of the subscription object in the format: **subs-***id*.
		
		Children of the subscription object include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).

Step 10	Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**).
	
	Example:
	```
	{
	      "telemetrySubscription": {
	        "attributes": {
	          "id": "30"
	        }
	        "children": [{
	          "telemetryRsSensorGroupRel":
	 {
	            "attributes": {
	              "sampleIntvl": "5000",
	             "tDn": "sys/tm/sensor-10"

	            }
	          }
	        }
	        ]
	      }
	    }
	```

Step 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.
	
	Example:
	```
	"telemetryRsSensorGroupRel": {
	    "attributes": {
	        "rType": "mo",
	        "rn":
	"rssensorGroupRel-[sys/tm/sensor-10]",
	        "sampleIntvl": "5000",
	        "tCl": "telemetrySensorGroup",
	        "tDn": "sys/tm/sensor-10",
	        "tType": "mo"
	```
		The relation object is of class **telemetryRsSensorGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		
		• **rn** — The relative name of the relation object in the format: **rssensorGroupRel-[sys/tm/***sensor-group-id***]**.
		
		• **sampleIntvl** — The data sampling period in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An

	Command or Action	Purpose

	```
	    }
	}
	```
		interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds.
		
		• **tCl** — The class of the target (sensor group) object, which is **telemetrySensorGroup**.
		
		• **tDn** — The distinguished name of the target (sensor group) object, which is **sys/tm/***sensor-group-id*.
		
		• **rType** — The relation type, which is **mo** for managed object.
		
		• **tType** — The target type, which is **mo** for managed object.

Step 12	Define one or more sensor paths or nodes to be monitored for telemetry.
	
	Example:
	
	Single sensor path
	```
	{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0",
	            "alias": "cdp_alias",
	        }
	    }
	}
	```
	Example:
	
	Single sensor path for NX-API
	```
	{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "show interface",
	            "path": "show bgp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	```
		A sensor path is defined in an object of class **telemetrySensorPath**. Configure the following attributes of the object:
		
		• **path** — The path to be monitored.
		
		• **rn** — The relative name of the path object in the format: **path-[***path***]**
		
		• **depth** — The retrieval level for the sensor path. A depth setting of **0** retrieves only the root MO properties.
		
		• **filterCondition** — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635
		
		• **alias** - Specify an alias for this path.

	Command or Action	Purpose

	```
	"filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}
	```
	 Example: Multiple sensor paths
	```
	{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	},
	{
	     "telemetrySensorPath": {
	         "attributes": {
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/dhcp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}
	```
	 Example: Single sensor path filtering for BGP disable events:
	```
	{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition":
	"eq(fmBgp.operSt.\"disabled\")",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	```

	Command or Action	Purpose
	`}`	
Step 13	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
Step 14	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
Step 15	Add the destination group object as a child object to the root element (**telemetryEntity**).	

Step 16	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
	
	Example:
	
	`"telemetryRtSensorGroupRel": {`
	`    "attributes": {`
	`        "rn":`
	`"rtsensorGroupRel-[sys/tm/subs-30]",`
	`       "tCl": "telemetrySubscription",`
	
	`        "tDn": "sys/tm/subs-30"`
	`    }`
	`}`
		The relation object is of class **telemetryRtSensorGroupRel** and is a child object of **telemetrySensorGroup**. Configure the following attributes of the relation object:
		
		• **rn** — The relative name of the relation object in the format: **rtsensorGroupRel-[sys/tm/***subscription-id***]**.
		
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 17	Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
	
	Example:
	
	`"telemetryRtDestGroupRel": {`
	`    "attributes": {`
	`        "rn":`
	`"rtdestGroupRel-[sys/tm/subs-30]",`
	`       "tCl": "telemetrySubscription",`
	
	`        "tDn": "sys/tm/subs-30"`
	`    }`
	`}`
		The relation object is of class **telemetryRtDestGroupRel** and is a child object of **telemetryDestGroup**. Configure the following attributes of the relation object:
		
		• **rn** — The relative name of the relation object in the format: **rtdestGroupRel-[sys/tm/***subscription-id***]**.
		
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 18	Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
	
	Example:
	
	`"telemetryRsDestGroupRel": {`
	`    "attributes": {`
	`        "rType": "mo",`
	`        "rn":`
		The relation object is of class **telemetryRsDestGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		
		• **rn** — The relative name of the relation object in the format: **rsdestGroupRel-[sys/tm/***destination-group-id***]**.

	Command or Action	Purpose

	`"rsdestGroupRel-[sys/tm/dest-20]",`
	`        "tCl": "telemetryDestGroup",`
	`        "tDn": "sys/tm/dest-20",`
	`        "tType": "mo"`
	`    }`
	`}`
		• **tCl** — The class of the target (destination group) object, which is **telemetryDestGroup**.
		• **tDn** — The distinguished name of the target (destination group) object, which is **sys/tm/***destination-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

Step 19	Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.	The base path for the telemetry entity is `sys/tm` and the NX-API endpoint is: `{{URL}}/api/node/mo/sys/tm.json`

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"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50051",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "5000",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

Configuration Example for Telemetry Using the NX-API

Streaming Paths to a Destination

This example creates a subscription that streams paths `sys/cdp` and `sys/ipv4` to a destination `1.2.3.4 port 50001` every five seconds.

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
                        "attributes": {
                            "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/cdp",
                            "rn": "path-[sys/cdp]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/ipv4",
                            "rn": "path-[sys/ipv4]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }]
            }
        }, {
            "telemetryDestGroup": {
                "attributes": {
                    "id": "20",
                    "rn": "dest-20"
                },
                "children": [{
                    "telemetryRtDestGroupRel": {
                        "attributes": {
                            "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetryDest": {
                        "attributes": {
                            "addr": "1.2.3.4",
                            "enc": "GPB",
                            "port": "50001",
                            "proto": "gRPC",
                            "rn": "addr-[1.2.3.4]-port-50001"
                        }
                    }
                }]
            }
        }, {
            "telemetrySubscription": {
                "attributes": {
                    "id": "30",
                    "rn": "subs-30"
                },
                "children": [{
                    "telemetryRsDestGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                            "tCl": "telemetryDestGroup",
                            "tDn": "sys/tm/dest-20",
                            "tType": "mo"
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

Filter Conditions on BGP Notifications

The following example payload enables notifications that trigger when the BFP feature is disabled as per the `filterCondition` attribute in the `telemetrySensorPath` MO. The data is streamed to`10.30.217.80 port 50055`.

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

Using Postman Collection for Telemetry Configuration

An example Postman collection is an easy way to start configuring the telemetry feature, and can run all telemetry CLI equivalents in a single payload. Modify the file in the preceding link using your preferred text editor to update the payload to your needs, then open the collection in Postman and run the collection.

Telemetry Model in the DME

The telemetry application is modeled in the DME with the following structure:

```
model
|----package [name:telemetry]
    |  @name:telemetry
    |----objects
         |----mo [name:Entity]
              |    @name:Entity
              |      @label:Telemetry System
              |--property
              |    @name:adminSt
              |      @type:AdminState
              |
              |----mo [name:SensorGroup]
              |    |    @name:SensorGroup
              |    |      @label:Sensor Group
              |     |--property
              |     |    @name:id [key]
              |     |      @type:string:Basic
              |     |    @name:dataSrc
              |     |      @type:DataSource
              |     |
              |     |----mo [name:SensorPath]
              |          |   @name:SensorPath
              |          |     @label:Sensor Path
              |          |--property
              |          |    @name:path [key]
              |          |      @type:string:Basic
              |          |    @name:filterCondition
              |          |      @type:string:Basic
              |          |    @name:excludeFilter
              |          |      @type:string:Basic
              |          |    @name:depth
              |          |      @type:RetrieveDepth
              |
              |----mo [name:DestGroup]
              |     |    @name:DestGroup
              |     |      @label:Destination Group
              |     |--property
              |     |    @name:id
              |     |      @type:string:Basic
              |     |
              |     |----mo [name:Dest]
              |          |   @name:Dest
              |          |     @label:Destination
              |          |--property
              |          |    @name:addr [key]
              |          |      @type:address:Ip
              |          |    @name:port [key]
              |          |      @type:scalar:Uint16
              |          |    @name:proto
              |          |      @type:Protocol
              |          |    @name:enc
              |          |      @type:Encoding
              |
              |----mo [name:Subscription]
                   |    @name:Subscription
                   |      @label:Subscription
                   |--property
                   |    @name:id
                   |      @type:scalar:Uint64
                   |----reldef
                   |     |  @name:SensorGroupRel
                   |     |    @to:SensorGroup
                   |     |    @cardinality:ntom
                   |     |    @label:Link to sensorGroup entry
                   |     |--property
                   |          @name:sampleIntvl
                   |            @type:scalar:Uint64
                   |
                   |----reldef
                        |  @name:DestGroupRel
                        |    @to:DestGroup
                        |    @cardinality:ntom
                        |    @label:Link to destGroup entry
```

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 302.

Note The model does not support counters for subinterface, loopback, or VLAN, so they are not streamed out.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

	Command or Action	Purpose

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**path interface**
	Example:
	`switch-1(conf-tm-sensor)# path interface`
	`switch-1(conf-tm-sensor)#`
		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.
		Depending on how the polling interval is configured, interface data is sent based on a periodic basis or whenever the interface state changes.

Step 5	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	Example:
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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**

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**path interface query-condition counters=[detailed]**
	Example:
	`switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed]`
	`switch-1(conf-tm-sensor)#`
		Configure the interface path label and query for only the nonzero counters from all interfaces.

Step 5	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	Example:
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

	Command or Action	Purpose

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

	Command or Action	Purpose

Step 3	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 4	**path interface query-condition oper-speed=[***speed***]**
	Example:
	`switch-1(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G]`
	`switch-1(conf-tm-sensor)#`
		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.

Step 5	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	Example:
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
	
	Example:
	`switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G]`
	`switch-1(conf-tm-sensor)#`
		Configures multiple conditions in the same query. In this example, the query does both of the following:
		• Gathers and returns non-zero counters on interfaces running at 1 Gbps.
		• Gathers and returns non-zero counters on interfaces running at 40 Gbps.

Step 5	**destination-group** *grp_id*
	
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	
	Example:
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

Step 7	**subscription** *sub_id*
	
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

	Command or Action	Purpose

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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 302.

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.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**path environment**
	Example:
	`switch-1(conf-tm-sensor)# path environment`
	`switch-1(conf-tm-sensor)#`
		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.

	Command or Action	Purpose
		Depending on the sample interval, the environment data is either streaming based on the polling interval, or sent when events occur.

Step 5	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)#`
	`destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	Example:
	`switch-1(conf-tm-dest)#` **`ip address`**
	`1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)#` **`subscription 33`**
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)#` **`snsr-grp 6`**
	`sample-interval 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 environment events occur.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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 302 .

This path does not support filtering.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1#` **`configure terminal`**
	`switch-1(config)#`
		Enter configuration mode.

	Command or Action	Purpose

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**path resources**
	Example:
	`switch-1(conf-tm-sensor)# path resources`
	`switch-1(conf-tm-sensor)#`
		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.
		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.

Step 5	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port*
	Example:
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(conf-tm-dest)#`
		Configure the telemetry data for the subscription to stream to the specified IP address and port.

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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 resource events occur.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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

	Command or Action	Purpose
		you configured in the **destination-group** 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 302.

This path does not support filtering.

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	
	Example:
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(conf-tm-sensor)#`
		Create a sensor group for telemetry data.

Step 4	**vxlan environment**
	
	Example:
	`switch-1(conf-tm-sensor)# vxlan environment`
	`switch-1(conf-tm-sensor)#`
		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	**destination-group** *grp_id*
	
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 6	**ip address** *ip_addr* **port** *port* **Example:**	Configure the telemetry data for the subscription to stream to the specified IP address and port.

	Command or Action	Purpose

	`switch-1(conf-tm-dest)# **ip address 1.2.3.4 port 50004**`
	`switch-1(conf-tm-dest)#`	

Step 7	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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 VXLAN events occur.

Step 9	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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.

Procedure

	Command or Action	Purpose
Step 1	**show running-config-telemetry** **Example:** `switch-1(conf-tm-sensor)# show running-config telemetry` (see output below)	Displays the current running config for telemetry, 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.

```
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]
```

	Command or Action	Purpose

	`subscription 6`
	`snsr-grp 6 sample-interval 6000`
	`nxosv2(conf-tm-sensor)#`	

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
show telemetry usability {all | environment | interface | resources | vxlan}	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. For the interfaces path label, also any keyword filters or query conditions you configured.
show running-config telemetry	The running configuration for telemetry and selected path information.

Command Examples

Note The **show telemetry usability all** command is a concatenation of all the individual commands that are shown in this section.

The following shows an example of the **show telemetry usability environment** command.

```
switch-1# show telemetry usability environment
  1) label_name           : environment

     path_name            : sys/ch
     query_type           : poll
     query_condition      :
rsp-subtree=full&query-target=subtree&target-subtree-class=eqptPsuSlot,eqptFtSlot,eqptSupCSlot,eqptPsu,eqptFt,eqptSensor,eqptLCSlot


  2) label_name           : environment

     path_name            : sys/ch
     query_type           : event
     query_condition      :
[illegible]
switch-1#
```

The following shows the output of the **show telemetry usability interface** command.

```
switch-1# show telemetry usability interface
  1) label_name           : interface
```

```
     path_name            : sys/intf
     query_type           : poll
     query_condition      :
query-target=children&query-target-filter=eq(l1PhysIf.adminSt,"up")&rsp-subtree=children&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut


  2) label_name           : interface

     path_name            : sys/mgmt-[mgmt0]
     query_type           : poll
     query_condition      :
query-target=subtree&query-target-filter=eq(mgmtMgmtIf.adminSt,"up")&rsp-subtree=full&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut


  3) label_name           : interface

     path_name            : sys/intf
     query_type           : event
     query_condition      :
query-target=subtree&query-target-filter=or(or(deleted(),created()),or(and(updated(ethpmPhysIf.operSt),eq(ethpmPhysIf.operSt,"down")),and(updated(ethpmPhysIf.operSt),eq(ethpmPhysIf.operSt,"up")),and(updated(ethpmRtdIf.operSt),eq(ethpmRtdIf.operSt,"down")),and(updated(ethpmRtdIf.operSt),eq(ethpmRtdIf.operSt,"up")),and(updated(ethpmAggrIf.operSt),eq(ethpmAggrIf.operSt,"down")),and(updated(ethpmAggrIf.operSt),eq(ethpmAggrIf.operSt,"up")),and(updated(ethpmEncRtdIf.operSt),eq(
ethpmEncRtdIf.operSt,"down")),and(updated(ethpmEncRtdIf.operSt),eq(ethpmEncRtdIf.operSt,"up"))))


  4) label_name           : interface

     path_name            : sys/mgmt-[mgmt0]
     query_type           : event
     query_condition      :
query-target=subtree&query-target-filter=or(or(deleted(),created()),or(and(updated(imMgmtIf.operSt),eq(imMgmtIf.operSt,"down")),and(updated(imMgmtIf.operSt),eq(imMgmtIf.operSt,"up"))))
switch-1#
```

The following shows an example of the **show telemetry usability resources** command.

```
switch-1# show telemetry usability resources
  1) label_name           : resources

     path_name            : sys/proc
     query_type           : poll
     query_condition      : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  2) label_name           : resources

     path_name            : sys/procsys
     query_type           : poll
     query_condition      :
query-target=subtree&target-subtree-class=procSysCore,procSysCpuSummary,procSysCpu,procIdle,procIrq,procKernel,procNice,procSoftirq,procTotal,procUser,procWait,procSysCpuHistory,procSysLoad,procSysMem,procSysMemFree,procSysMemUsage,procSysMemUsed


  3) label_name           : resources

     path_name            : sys/procsys/sysmem
     query_type           : event
     query_condition      :
query-target-filter=and(updated(procSysMem.memstatus),ne(procSysMem.memstatus,"OK"))

switch-1#
```

The following shows an example of the **show telemetry usability vxlan** command.

```
switch-1# show telemetry usability vxlan
  1) label_name           : vxlan

     path_name            : sys/bd
     query_type           : poll
     query_condition      : query-target=subtree&target-subtree-class=l2VlanStats
  2) label_name          : vxlan

     path_name           : sys/eps
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  3) label_name          : vxlan

     path_name           : sys/eps
     query_type          : event
     query_condition     : query-target=subtree&target-subtree-class=nvoDyPeer

  4) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
    query_condition     : query-target=subtree&query-target-filter=or(deleted(),created())


  5) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
     query_condition     :
query-target=subtree&target-subtree-class=bgpDom,bgpPeer,bgpPeerAf,bgpDomAf,bgpPeerAfEntry,bgpOperRtctrlL3,bgpOperRttP,bgpOperRttEntry,bgpOperAfCtrl

switch-1#
```

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	Sends event notifications for create, update, and delete events. The following values are exported through telemetry for the RIB path: • Next-hop routing information: • Address of the next hop • Outgoing interface for the next hop • VRF name for the next hop • Owner of the next hop • Preference for the next hop • Metric for the next hop • Tag for the next hop • Segment ID for the next hop • Tunnel ID for the next hop • Encapsulation type for the next hop • Bitwise OR of flags for the Next Hop Type • For Layer-3 routing information: • VRF name of the route • Route prefix address • Mask length for the route • Number of next hops for the route • Event type • Next hops

Path Type	Subscription Baseline	Event Notifications
MAC	Executes a `GETALL` from DME for static and dynamic MAC entries	Sends event notifications for add, update, and delete events. The following values are exported through telemetry for the MAC path: • MAC address • MAC address type • VLAN number • Interface name • Event types Both static and dynamic entires are supported in event notifications.
Adjacency	Sends the IPv4 and IPv6 adjacencies	Sends event notifications for add, update, and delete events. The following values are exported through telemetry for the Adjacency path: • IP address • MAC address • Interface name • Physical interface name • VRF name • Preference • Source for the adjacency • Address family for the adjacency • Adjacency event type

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**.

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 317.

Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		Create a sensor group.

Step 4	**data-source native**
	Example:
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(conf-tm-sensor)#`
		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	**path rib**
	Example:
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
		Configure the RIB path which streams routes and route update information.

Step 6	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

	Command or Action	Purpose

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`
		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	**subscription** *sub_id*
	
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 10	**dst-group** *dgrp_id*
	
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** command.

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 317.

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**).

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		Create a sensor group.

Step 4	**data-source native**
	Example:
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(conf-tm-sensor)#`
		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	**path mac**
	Example:
	`nxosv2(conf-tm-sensor)# path mac`
	`nxosv2(conf-tm-sensor)#`
		Configure the MAC path which streams information about MAC entries and MAC notifications.

Step 6	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	Example:
		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`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`	

Step 8	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 10	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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 317.

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**).

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:**	Enter configuration mode.

	Command or Action	Purpose

	`switch-1# configure terminal`
	`switch-1(config)#`	

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		Create a sensor group.

Step 4	**data-source native**
	Example:
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(conf-tm-sensor)#`
		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	**path mac-all**
	Example:
	`nxosv2(conf-tm-sensor)# path mac-all`
	`nxosv2(conf-tm-sensor)#`
		Configure the MAC path which streams information about all MAC entries and MAC notifications.

Step 6	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`
		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

Step 8	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 10	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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 317.

Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		Create a sensor group.

	Command or Action	Purpose

Step 4	**data-source native**
	Example:
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(conf-tm-sensor)#`
		Set the data source to native so that any native application can use the streamed data.

Step 5	**path adjacency**
	Example:
	`nxosv2(conf-tm-sensor)# path adjacency`
	`nxosv2(conf-tm-sensor)#`
		Configure the Adjacency path which streams information about the IPv4 and IPv6 adjacencies.

Step 6	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`
		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	**subscription** *sub_id*
	Example:
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`
		Enter telemetry subscription submode, and configure the telemetry subscription.

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

	Command or Action	Purpose

Step 10	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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 317.

Before you begin

If you have not enabled the telemetry feature, enable it now (**feature telemetry**).

Procedure

	Command or Action	Purpose

Step 1	**configure terminal**
	Example:
	`switch-1# configure terminal`
	`switch-1(config)#`
		Enter configuration mode.

	Command or Action	Purpose

Step 2	**telemetry**
	Example:
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		Enter configuration mode for the telemetry features.

Step 3	**sensor-group** *sgrp_id*
	Example:
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		Create a sensor group.

Step 4	**data-source native**
	Example:
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(conf-tm-sensor)#`
		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	**path rib**
	Example:
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
		Configure the RIB path which streams routes and route update information.

Step 6	**destination-group** *grp_id*
	Example:
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(conf-tm-dest)#`
		Enter telemetry destination group submode and configure the destination group.

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	Example:
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`
		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	**subscription** *sub_id* **Example:**	Enter telemetry subscription submode, and configure the telemetry subscription.

	Command or Action	Purpose

	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(conf-tm-sub)#`	

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	Example:
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 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.

Step 10	**dst-group** *dgrp_id*
	Example:
	`switch-1(conf-tm-sub)# dst-grp 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 **destination-group** 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: • message-id • node-name • time-stamp • time-of-day • time-zone • category • message-name • severity • text

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.	*Telemetry Deployment for VXLAN EVPN Solution*

PART V

XML Management Interface

• XML Management Interface, on page 337

CHAPTER 24

XML Management Interface

This section contains the following topics:

• About the XML Management Interface, on page 337
• Licensing Requirements for the XML Management Interface, on page 338
• Prerequisites to Using the XML Management Interface, on page 339
• Using the XML Management Interface, on page 339
• Information About Example XML Instances, on page 351
• Additional References, on page 357

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 341 and RFC 4741.

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

This section includes the following topics:

• NETCONF Layers, on page 337
• SSH xmlagent, on page 338

NETCONF Layers

The following are the NETCONF layers:

Table 19: 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 341 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 *Cisco NX-OS Licensing Guide*.

Prerequisites to Using the XML Management Interface

The XML management interface has the following prerequisites:

- You must install SSHv2 on the client PC.
- You must install an XML management tool that supports NETCONF over SSH on the client PC.
- You must set the appropriate options for the XML server on the device.

Using the XML Management Interface

This section describes how to manually configure and use the XML management interface. Use the XML management interface with the default settings on the device.

Configuring SSH and the XML Server Options

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 20: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 351
commit	Sets the running configuration to the current contents of the candidate configuration.	NETCONF Commit Instance - Candidate Configuration Capability, on page 356
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.	NETCONF Confirmed-commit Instance , on page 356
copy-config	Copies the content of source configuration datastore to the target datastore.	NETCONF copy-config Instance, on page 352
delete-config	Operation not supported.	—
edit-config	Configures features in the running configuration of the device. You use this operation for configuration commands.	NETCONF edit-config Instance, on page 352 NETCONF rollback-on-error Instance , on page 356
get	Receives configuration information from the device. You use this operation for **show** commands. The source of the data is the running configuration.	Creating NETCONF XML Instances, on page 341
get-config	Retrieves all or part of a configuration	NETCONF get-config Instance, on page 354
kill-session	Closes the specified XML server session. You cannot close your own session. See the close-session NETCONF operation.	NETCONF Kill-session Instance, on page 352

NETCONF Operation	Description	Example
lock	Allows the client to lock the configuration system of a device.	NETCONF Lock Instance, on page 354
unlock	Releases the configuration lock that the session issued.	NETCONF unlock Instance, on page 355
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	NETCONF validate Capability Instance , on page 357

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 340 section.

Using this schema, it is possible to build an XML instance. In the following examples, the relevant portions of the nfcli.xsd schema file that was used to build Creating NETCONF XML Instances, on page 341 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 350 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 21: 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 *Configuration CLI Commands Sent Through <exec-command>*.

Replies to configuration commands that are sent through the <cmd> tag are as follows:

- <nf:ok>: All configure commands are executed successfully.
- <nf:rpc-error>: Some commands have failed. The operation stops on the first error, and the <nf:rpc-error> subtree provides more information on what configuration failed. Notice that any configuration 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>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

NETCONF Replies

For every XML request sent by the client, the XML server sends an XML response enclosed in the RPC response tag <rpc-reply>.

This section contains the following topics:

• RPC Response Tag, on page 350
• Interpreting Tags Encapsulated in the Data Tag, on page 350

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 22: RPC Response Elements

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

Interpreting Tags Encapsulated in the Data Tag

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

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 351
- NETCONF Kill-session Instance, on page 352
- NETCONF copy-config Instance, on page 352
- NETCONF edit-config Instance, on page 352
- NETCONF get-config Instance, on page 354
- NETCONF Lock Instance, on page 354
- NETCONF unlock Instance, on page 355
- NETCONF Commit Instance - Candidate Configuration Capability, on page 356
- NETCONF Confirmed-commit Instance , on page 356
- NETCONF rollback-on-error Instance , on page 356
- NETCONF validate Capability Instance , on page 357

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

RFCs	Title
RFC 4741	NETCONF Configuration Protocol
RFC 4742	Using the NETCONF Configuration Protocol over Secure Shell (SSH)