Cisco Nexus 3000 Series NX-OS Programmability Guide, Release 7.x

First Published: 2015-05-22

Last Modified: 2020-03-16

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

CONTENTS

Preface

This preface includes the following sections:

Audience

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

Document Conventions

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
Italic	Italic text indicates arguments for which 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 3000 Series Switches

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

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

Documentation Feedback

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

Communications, Services, and Additional Information

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

Cisco Bug Search Tool

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

CHAPTER 1

New and Changed Information

This chapter contains the following sections:

New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 3000 Series NX-OS Programmability Guide, Release 7.x* and where they are documented.

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
Model-Driven Telemetry	Enhanced **show telemetry control database sensor-paths** to show details about encoding, collection, transport, and streaming.	7.0(3)I7(7)	Displaying Telemetry Configuration and Statistics
XMLIN	Support for converting NX-OS CLI commands to Network Configuration format is documented.	7.0(3)I7(4)	Converting CLI Commands to Network Configuration Format, on page 187
XML Management Interface	Support for managing the Cisco Nexus 3000 switches with an XML-based tool through the XML-based Network Configuration Protocol (NETCONF) is documented.	7.0(3)I7(4)	XML Management Interface, on page 255
JSON-RPC as an NX-API input request format	JSON-RPC is now an accepted NX-API input request format.	7.0(3)I7(4)	NX-API CLI, on page 133

Feature	Description	Changed in Release	Where Documented
rollback NX-API request element	The **rollback** NX-API request element is now available in XML, JSON and JSON-RPC input request formats.	7.0(3)I7(4)	NX-API CLI, on page 133
Puppet support for Cisco Nexus 3500 devices.	Added Puppet support for Cisco Nexus 3500 devices.	NX-OS 7.0(3)I7(2)	Prerequisites
Chef support for Cisco Nexus 3500 devices.	Added Chef support for Cisco Nexus 3500 devices.	NX-OS 7.0(3)I7(2)	Prerequisites
Guest Shell support for Cisco Nexus 3500 devices.	Added Guest Shell support for Cisco Nexus 3500 devices.	NX-OS 7.0(3)I7(1)	About the Guest Shell, on page 23
Export rootfs of Guest Shell onto multiple devices.	Support to export a specific Guest Shell rootfs and deploy it onto multiple devices.	7.0(3)I7(1)	Replicating the Guest Shell , on page 44
Telemetry support for UDP and secure UDP (DTLS) protocols	Added telemetry support for UDP and secure UDP (DTLS) protocols	7.0(3)I7(1)	Telemetry Components and Process, on page 217
Authentication with self signed SSL certificate	Added support for authentication with self signed SSL certificate.	7.0(3)I7(1)	Guidelines and Limitations for Telemetry, on page 220
Telemetry VRF Support	Added telemetry VRF support.	7.0(3)I7(1)	Telemetry VRF Support, on page 223
Telemetry Compression for gRPC Transport	Added support for telemetry compression for gRPC transport.	7.0(3)I7(1)	Telemetry Compression for gRPC Transport, on page 221
Model-Driven Telemetry	Added support for telemetry.	7.0(3)I7(1)	Model-Driven Telemetry, on page 217
NX-SDK	NX-SDK (C++ abstraction/plugin library layer) available.	7.0(3)I7(1)	About the NX-SDK, on page 123
NX-OS Programmable Interface Component RPM packages	NX-OS Programmable Interface Component RPM packages included in the NX-OS image.	7.0(3)I7(1)	About the Component RPM Packages, on page 173
Dynamic Logging	Users can now update configurations without restarting the switch.	7.0(3)I6(1)	Dynamic Logger, on page 209

Feature	Description	Changed in Release	Where Documented
Guest Shell	Version updated from 2.1 to 2.2. Guest shell now includes the following features: • User accounts will have the same name as the one used to log into the switch, • The dohost utility will send the logged-in user name over the NX-API connection, • A network administrator can configure non-administrative accounts within the guest shell.	7.0(3)I5(2)	Guest Shell, on page 23
NX-API Developer Sandbox	The Sandbox now supports the generation of YANG and NX-API REST payloads.	7.0(3)I5(1)	NX-API Developer Sandbox, on page 155
Model-Driven Programmability	Data modeling provides a programmatic and standards-based method of writing configurations to the network device.	7.0(3)I5(1)	Model-Driven Programmability, on page 167
XML Management Interface	Added the use of XML Management Interface to configure devices.	7.0(3)I4(1)	Replaced in a later release by Model-Driven Programmability
NX-API REST	Added support for the Cisco Nexus 3132Q-XL, 3172PQ-XL, and 3172TQ-XL switches.	7.0(3)I2(2)	NX-API REST
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 Cisco Nexus switch.	7.0(3)i2(1)	Nexus Application Development - ISO, on page 111

Feature	Description	Changed in Release	Where Documented
Puppet	The Puppet software package, developed by Puppet Labs, is an open source automation toolset for managing servers and other resources by enforcing device states, such as configuration settings.	7.0(3)i2(1)	Puppet Agent
Third-Party Applications	Open source applications added.	7.0(3)i2(1)	Third-Party Applications, on page 85
Guest Shell 2.0	Updated to Guest Shell 2.0	7.0(3)i2(1)	Guest Shell, on page 23
iPXE	Open source network boot firmware.	7.0(3)i2(1)	iPXE, on page 75
Kernel Stack	Uses well known Linux APIs to manage the routes and front panel ports.	7.0(3)i2(1)	Kernel Stack, on page 79
Broadcom Shell	The Cisco Nexus 3000 Series device front panel and fabric module line cards contain Broadcom Network Forwarding Engines (NFE). The number of NFEs varies depending upon the specific model of the front panel line card (LC) or the fabric module (FM).	7.0(3)i2(1)	Broadcom Shell, on page 55
No updates since release 6.x		7.0(3)I1(1)	

CHAPTER 2

Overview

Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 3000 Series devices is as follows:

- **Resilient**

  Provides critical business-class availability.

- **Modular**

  Has extensions that accommodate business needs.

- **Highly Programmatic**

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

- **Secure**

  Protects and preserves data and operations.

- **Flexible**

  Integrates and enables new technologies.

- **Scalable**

  Accommodates and grows with the business and its requirements.

- **Easy to use**

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

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

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

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

Standard Network Manageability Features

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

Advanced Automation Feature

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

PowerOn Auto Provisioning Support

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

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

OpenStack Integration

The Cisco Nexus 3000 Series devices support the Cisco Nexus plugin for OpenStack Networking, also known as Neutron (http://www.cisco.com/web/solutions/openstack/index.html). The plugin allows you to build an infrastructure as a service (IaaS) network and to deploy a cloud network. With OpenStack, you can build an on-demand, self-service, multitenant computing infrastructure. However, implementing OpenStack's VLAN networking model across virtual and physical infrastructures can be difficult.

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

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

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

Note We recommend configuring the REST API driver. For more configuration details, go to:

http://docwiki.cisco.com/wiki/Neutron_ML2_Driver_For_Cisco_Nexus_Devices_Ocata_Release

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

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

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

Programmability Support

Cisco NX-OS on Cisco Nexus 3000 devices support the following capabilities to aid programmability:

NX-API Support

Cisco NX-API allows for HTTP-based programmatic access to the Cisco Nexus 3000 platform. This support is delivered by NX-API, an open source webserver. NX-API provides the configuration and management capabilities of the Cisco NX-OS CLI with web-based APIs. The device can be set to publish the output of the API calls in XML or JSON format. This API enables rapid development on the Cisco Nexus 3000 platform.

Python Scripting

Cisco Nexus 3000 devices support Python v2.7.5 in both interactive and non-interactive (script) modes.

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

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

Tcl Scripting

Cisco Nexus 3000 Series devices support tcl (Tool Command Language). Tcl is a scripting language that enables greater flexibility with CLI commands on the switch. You can use tcl to extract certain values in the output of a **show** command, perform switch configurations, run Cisco NX-OS commands in a loop, or define EEM policies in a script.

Broadcom Shell

The Cisco Nexus 3000 Series device front panel and fabric module line cards contain Broadcom Network Forwarding Engine (NFE). You can access the Broadcom command line shell (bcm-shell) from these NFEs.

Bash

Cisco Nexus 3000 devices support direct Bourne-Again SHell (Bash) access. With Bash, you can access the underlying Linux system on the device and manage the system.

Guest Shell

The Cisco Nexus 3000 Series devices support a guest shell that provides Bash access into a Linux execution space on the host system that is decoupled from the host Cisco Nexus 3000 NX-OS software. With the guest shell you can add software packages and update libraries as needed without impacting the host system software.

Container Tracker Support

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

The following commands communicate with the Kubernetes API Server:

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

PART I

Shells and Scripting

CHAPTER 3

Bash

About Bash

In addition to the NX-OS CLI, Cisco Nexus 3000 Series devices support access to the Bourne-Again SHell (Bash). Bash interprets commands that you enter or commands that are read from a shell script. Using Bash enables access to the underlying Linux system on the device and to manage the system.

Guidelines and Limitations

The Bash shell has the following guidelines and limitations:

- When importing Cisco Python modules, do not use Python from the Bash shell. Instead use the more recent Python in NX-OS VSH.

Accessing Bash

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

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

```
switch# show role name dev-ops

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

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

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

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

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

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

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

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

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

```
run bash whoami
```

Escalate Privileges to Root

The privileges of an admin user can escalate their privileges for root access. Root access is required to pass configuration commands to the NX-OS VSH.

The following are guidelines for escalating privileges:

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

- SSH to the switch using `root` username through a non-management interface will default to Linux Bash shell-type access for the root user. Type **vsh** to return to NX-OS shell access.

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

```
switch# run bash
Linux# sudo su root

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

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

Password:

Linux# whoami
root
Linux# exit
exit
```

Examples of Bash Commands

This section contains examples of Bash commands and output.

Displaying System Statistics

The following example shows how to display system statistics:

```
switch# run bash
Linux# cat /proc/meminfo
MemTotal:      3795100 kB
MemFree:       1472680 kB
Buffers:           136 kB
Cached:        1100116 kB
ShmFS:         1100116 kB
Allowed:        948775 Pages
Free:           368170 Pages
Available:      371677 Pages
SwapCached:          0 kB
Active:        1198872 kB
Inactive:       789764 kB
SwapTotal:           0 kB
SwapFree:            0 kB
Dirty:               0 kB
Writeback:           0 kB
AnonPages:      888272 kB
Mapped:         144044 kB
Slab:           148836 kB
SReclaimable:    13892 kB
SUnreclaim:     134944 kB
PageTables:      28724 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
WritebackTmp:        0 kB
CommitLimit:   1897548 kB
```

```
Committed_AS: 19984932 kB
VmallocTotal: 34359738367 kB
VmallocUsed:    215620 kB
VmallocChunk: 34359522555 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
HugePages_Surp:      0
Hugepagesize:     2048 kB
DirectMap4k:     40960 kB
DirectMap2M:   4190208 kB
Linux#
```

Running Bash from CLI

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

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

...

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

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.

	Command or Action	Purpose
Step 2	**sudo 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$ sudo 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$ sudo yum list available
bgp.lib32_n9000                         1.0.0-r0
bash-4.2$ sudo yum -y install bfd
```

Upgrading Feature RPMs

Before you begin

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

Procedure

	Command or Action	Purpose
Step 1	**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 a Feature RPM

Procedure

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

Example

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

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

Erasing a Feature RPM

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

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

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

Procedure

	Command or Action	Purpose
Step 1	**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 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#

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

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, the Cisco Nexus 3000 Series devices support access to a decoupled execution space running within a Linux Container (LXC) called the "Guest Shell".

From within the Guest Shell the network-admin has the following capabilities:

- Access to the network over Linux network interfaces.
- Access to Cisco Nexus 3000 bootflash.
- Access to Cisco Nexus 3000 volatile tmpfs.
- Access to Cisco Nexus 3000 CLI.
- Access to Cisco NX-API REST.
- The ability to install and run python scripts.
- The ability to install and run 32-bit and 64-bit Linux applications.

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

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

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

Note Beginning with NX-OS 7.0(3)I7(1), the Guest Shell is supported on the Cisco Nexus 3500 switch.

Guidelines and Limitations

The Guest Shell has the following guideline and limitations:

Common Guidelines Across All Releases

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

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

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

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

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

Guest Shell installations prior to 2.2 (0.2) with Cisco Nexus release 7.0(3)I5(2) will not dynamically create individual user accounts.

- Installing the Cisco Nexus series switch software release on a fresh out-of-the-box Cisco Nexus switch will automatically enable the Guest Shell. Subsequent upgrades to the Cisco Nexus series switch software will NOT automatically upgrade Guest Shell.
- Guest Shell releases increment the major number when distributions or distribution versions change.
- Guest Shell releases increment the minor number when CVEs have been addressed. The Guest Shell will update CVEs only when CentOS makes them publically available.
- Cisco recommends using **yum update** to pick up third-party security vulnerability fixes directly from the CentOS repository. This provides the flexibility of getting updates as, and when, available without needing to wait for a Cisco NX-OS software update.

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

Upgrading from Guest Shell 1.0 to Guest Shell 2.x

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

Guest Shell 2.x

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

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

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

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

```
switch#
Installer will perform compatibility check first. Please wait.
```

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

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

Pre-Configured SSHD Service

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

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

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

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

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

1. Create the following files: `/usr/lib/systemd/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, the Guest Shell is accessible to the network-admin. It is automatically enabled in the system and can be accessed using the **run guestshell** command. Consistent with the **run bash** command, these commands can be issued within the Guest Shell with the **run guestshell** *command* form of the NX-OS CLI command.

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

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

Note When running in the Guest Shell, you have network-admin level privileges.

Note 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	256MB	256/3840MB
Storage	200MB	200/2000MB

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

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

Capabilities in the Guest Shell

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

The Guest Shell is populated with CentOS 7 Linux which provides the ability to Yum install software packages built for this distribution. The Guest Shell 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. Python 2.7.5 is included by default as is the PIP for installing additional python packages.

By default the Guest Shell is a 64-bit execution space. If 32-bit support is needed, the glibc.i686 package can be Yum installed.

The Guest Shell has access to the Linux network interfaces used to represent the management and data ports of the switch. Typical Linux methods and utilities like **ifconfig** and **ethtool** can be used to collect counters. When an interface is placed into a VRF in the NX-OS CLI, the Linux network interface is placed into a network namespace for that VRF. The name spaces can be seen at `/var/run/netns` and the **ip netns** utility can be used to run in the context of different namespaces. A couple of utilities, **chvrf** and **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 7 environments, including the Guest Shell.

NX-OS CLI in the Guest Shell

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

When invoking the **dohost** command each NX-OS command may be in single or double quotes:

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

The NX-OS CLI can be chained together:

```
[guestshell@guestshell ~]$ dohost "sh lldp time | in Hold" "show cdp global"
Holdtime in seconds: 120
Global CDP information:
CDP enabled globally
Refresh time is 21 seconds
Hold time is 180 seconds
CDPv2 advertisements is enabled
DeviceID TLV in System-Name(Default) Format
[guestshell@guestshell ~]$
```

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

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

Note For release 7.0(3)I5(2) using 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 Cisco Nexus 3000 device is in a network that uses an HTTP proxy server, the **http_proxy** and **https_proxy** environment variables must be set up within the Guest Shell also.

Example:

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

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

Access to Bootflash in Guest Shell

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

Example:

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

Python in Guest Shell

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

Example:

```
guestshell:~$ python
Python 2.7.5 (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 2.x (Centos 7)

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

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

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

```
[admin@guestshell ~]$ sudo su
[root@guestshell admin]# yum install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
  Installing : scl-utils-20130529-19.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, go to the CentOS 7 repo at http://mirror.centos.org/centos/7/os/x86_64/Packages/.

Yum resolves the dependancies 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 Virtual Services

Use of the Guest Shell and virtual services in Cisco Nexus 3000 series devices are only two of the many ways that the network-admin can manage or extend the functionality of the system. These options are geared towards providing an execution environment that is decoupled from the native host context. This separation allows the introduction of software into the system that may not be compatible with the native execution environment. It also allows the software to run in an environment that does not interfere with the behavior, performance, or scale of the system.

Digitally Signed Application Packages

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

The Cisco Nexus 3000 Series switches support the configuration of a signing level policy to allow for unsigned OVA software packages. To allow unsigned and Cisco-signed packages for creating virtual-services, the network-admin can configure the following:

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

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

Kernel Vulnerability Patches

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

ASLR and X-Space Support

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

Root-User Restrictions

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

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

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

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

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

Resource Management

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

Guest File System Access Restrictions

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

`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 3: 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.
guestshell export rootfs package *destination-file-URI*	Exports a Guest Shell **rootfs** file to a local URI (bootflash, USB1, etc.). (7.0(3)I7(1) and later releases)

Commands	Description
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.
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.

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

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

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

Disabling the Guest Shell

The **guestshell disable** command shuts down and disables the Guest Shell.

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

Example:

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

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

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

Destroying the Guest Shell

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

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

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

switch# guestshell destroy
```

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

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

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

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

Enabling the Guest Shell

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

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

Example:

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

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

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

Enabling the Guest Shell on Cisco Nexus 3000 with Compacted Image

The Guest Shell software is not available in a Cisco Nexus system image that has been compacted for the Cisco Nexus 3000 Series platforms with 1.6 GB bootflash and 4 GB RAM. You can still use the Guest Shell in this case, but you will need to download the software package from software.cisco.com for the Cisco Nexus release, then you will need to copy it onto the Cisco Nexus 3000 system and enable it.

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

The Guest Shell software installs onto the bootflash of the Cisco Nexus system. To create as much free bootflash space as possible, put the downloaded `guestshell.ova` file onto the `volatile:` storage media. Once the Guest Shell is successfully activated, the `guestshell.ova` file can be deleted. It will not be needed again unless the Guest Shell is destroyed at some point and needs to be re-installed.

For example:

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

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

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

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

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

Replicating the Guest Shell

Beginning with Cisco NX-OS release 7.0(3)I7(1), a Guest Shell **rootfs** that is customized on one switch can be deployed onto multiple switches.

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

Exporting Guest Shell rootfs

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

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

The **guestshell export rootfs package** command:

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

Importing Guest Shell rootfs

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

- Use the **guestshell enable package** *rootfs-file-URI* command to import a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.
- Use the **guestshell upgrade package** *rootfs-file-URI* command to import a Guest Shell **rootfs** when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.

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

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

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

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

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

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

Note To restore the embedded version of the rootfs:

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

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

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

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

Example workflow for **guestshell enable** :

```
switch# copy scp://user@10.1.1.1/my_storage/gs_rootfs.ext4 bootflash: vrf management
switch# guestshell resize cpu 8
Note: System CPU share will be resized on Guest shell enable
switch# guestshell enable package bootflash:gs_rootfs.ext4
Validating the provided rootfs
switch# 2017 Jul 31 14:58:01 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual
service 'guestshell+'
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
 'guestshell+'
2017 Jul 31 14:58:33 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```

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

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

Importing YAML File

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

Example of a Guest Shell import YAML file:

```
---
import-schema-version: "1.0"
info:
  name: "GuestShell"
  version: "2.2(0.3)"
  description: "Exported GuestShell: 20170216T175137Z"
app:
  apptype: "lxc"
  cpuarch: "x86_64"
  resources:
    cpu: 3
    memory: 307200
    disk:
```

```
    - target-dir: "/"
      capacity: 250
...
```

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

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

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

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

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

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

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

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

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Guest Shell import schema",
  "description": "Schema for Guest Shell import descriptor file - ver 1.0",
  "copyright": "2017 by Cisco systems, Inc. All rights reserved.",
  "id": "",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "import-schema-version": {
      "id": "/import-schema-version",
      "type": "string",
      "minLength": 1,
      "maxLength": 20,
      "enum": [
          "1.0"
      ]
    },
    "info": {
      "id": "/info",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "id": "/info/name",
          "type": "string",
          "minLength": 1,
          "maxLength": 29
        },
        "description": {
          "id": "/info/description",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "version": {
          "id": "/info/version",
          "type": "string",
          "minLength": 1,
          "maxLength": 63
        },
        "author-name": {
          "id": "/info/author-name",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "author-link": {
          "id": "/info/author-link",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        }
      }
    },
    "app": {
      "id": "/app",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "apptype": {
          "id": "/app/apptype",
          "type": "string",
          "minLength": 1,
          "maxLength": 63,
          "enum": [
            "lxc"
          ]
        },
        "cpuarch": {
          "id": "/app/cpuarch",
          "type": "string",
          "minLength": 1,
          "maxLength": 63,
          "enum": [
            "x86_64"
          ]
        },
        "resources": {
          "id": "/app/resources",
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "cpu": {
              "id": "/app/resources/cpu",
              "type": "integer",
              "multipleOf": 1,
              "maximum": 100,
              "minimum": 1
            },
            "memory": {
              "id": "/app/resources/memory",
              "type": "integer",
              "multipleOf": 1024,
              "minimum": 1024
            },
            "disk": {
              "id": "/app/resources/disk",
              "type": "array",
              "minItems": 1,
              "maxItems": 1,
              "uniqueItems": true,
              "items": {
                "id": "/app/resources/disk/0",
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "target-dir": {
                    "id": "/app/resources/disk/0/target-dir",
                    "type": "string",
                    "minLength": 1,
                    "maxLength": 1,
                    "enum": [
                      "/"
                    ]
                  },
                  "file": {
                    "id": "/app/resources/disk/0/file",
                    "type": "string",
                    "minLength": 1,
                    "maxLength": 63
                  },
                  "capacity": {
                    "id": "/app/resources/disk/0/capacity",
                    "type": "integer",
                      "multipleOf": 1,
                      "minimum": 1
                  }
                }
              }
            }
          },
          "required": [
            "memory",
            "disk"
          ]
        }
      },
      "required": [
        "apptype",
        "cpuarch",
        "resources"
      ]
    }
  },
  "required": [
    "app"
  ]
}
```

show guestshell Command

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

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

```
switch# show guestshell detail
Virtual service guestshell+ detail
  State                 : Activated
  Package information
    Name                : 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 : 1.9`
`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.

Command	Description

show virtual-service list
```
switch# **show virtual-service list ***

Virtual Service List:

Name                    Status         Package Name
-----------------------------------------------------------------
guestshell+             Activated      guestshell.ova
chef                  Installed     chef-0.8.1-n9000-spa-k9.ova
```
	Displays a summary of the virtual services, the status of the virtual services, and installed software packages.

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                : 250 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]#
```

CHAPTER 5

Broadcom Shell

About the Broadcom Shell

The Cisco Nexus 3000 Series device front panel and fabric module line cards contain Broadcom Network Forwarding Engines (NFE). The number of NFEs varies depending upon the specific model of the front panel line card (LC) or the fabric module (FM).

Guidelines and Limitations

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

Accessing the Broadcom Shell (bcm-shell)

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

Accessing bcm-shell with the CLI API

The bcm-shell commands are passed directly from the Cisco Nexus 3000 Series CLI to the specific T2 ASIC instance. The T2 ASIC instance can be on the fabric module or on the front panel line card.

The command syntax is as follows:

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

Where

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

command	Broadcom command

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

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

Accessing the Native bcm-shell on the Fabric Module

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

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

- Use the **show module** command to display the FMs.

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

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

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

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

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

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

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

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

Accessing the bcm-shell on the Line Card

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

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

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

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

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

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

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

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

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

CHAPTER 6

Python API

About the Python API

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

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

http://www.python.org/

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

The Cisco Nexus 3000 Series devices 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.

For more Python modules, you can install the `python-modules-nxos` RPM (`python-modules-nxos-1.0.0-9.2.1.lib32_x86.rpm`) from https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86_64/. Refer to the "Manage Feature RPMs" section for instructions on installing an RPM.

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

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

NAME
    cisco

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

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

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

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 4: 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.

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 from the CLI:

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

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

Display Formats

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

Example 1:

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

Example 2:

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

Example 3:

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

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

Example 4:

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                kern_uptm_secs = 21
                kick_file_name = bootflash:///nxos.9.2.1.bin.S246
                    rr_service = None
                     module_id = Supervisor Module
                   kick_tmstmp = 07/11/2018 00:01:44
                bios_cmpl_time = 05/17/2018
                bootflash_size = 20971520
             kickstart_ver_str = 9.2(1)
                kick_cmpl_time = 7/9/2018 9:00:00
                    chassis_id = Nexus9000 C9504 (4 Slot) Chassis
                 proc_board_id = SAL171211LX
                        memory = 16077872
                  manufacturer = Cisco Systems, Inc.
                kern_uptm_mins = 26
                  bios_ver_str = 05.31
                      cpu_name = Intel(R) Xeon(R) CPU D-1528 @ 1.90GHz
                 kern_uptm_hrs = 2
                      rr_usecs = 816550
                    rr_sys_ver = 9.2(1)
                     rr_reason = Reset Requested by CLI command reload
                      rr_ctime = Wed Jul 11 20:44:39 2018
                    header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2018, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.
                     host_name = switch
                      mem_type = kB
                kern_uptm_days = 0
>>>
```

Non-interactive Python

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

The Cisco Nexus 3000 Series device also supports the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

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

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

from cli import *
import sys, time

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

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

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

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

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

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

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

Running Scripts with Embedded Event Manager

On Cisco Nexus 3000 Series devices, embedded event manager (EEM) policies support Python scripts.

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

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

```
switch# show running-config eem

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

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

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

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

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

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

PC_VSH_CMD_TLV(7679) with q
```

Python Integration with Cisco NX-OS Network Interfaces

On Cisco Nexus 3000 Series devices, Python is integrated with the underlying Cisco NX-OS network interfaces. You can switch from one virtual routing context to another by setting up a context through the cisco.vrf.set_global_vrf() API.

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

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

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

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

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

    Returns: Nothing

>>>
```

Cisco NX-OS Security with Python

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

All users 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 nonprivileged 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 nonprivileged user:

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

The following example shows an RBAC configuration:

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

Example of Running Script with Scheduler

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

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

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

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

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

CHAPTER 7

Scripting with Tcl

About Tcl

Tcl (pronounced "tickle") is a scripting language that increases flexibility of CLI commands. You can use Tcl to extract certain values in the output of a **show** command, perform switch configurations, run Cisco NX-OS commands in a loop, or define Embedded Event Manager (EEM) policies in a script.

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

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 8

iPXE

This chapter contains the following sections:

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 more protocols.

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

- Boots from a web server via HTTP, iSCSI SAN, FCoE, and so on
- Supports both IPv4 and IPv6
- Netboot supports HTTP/TFTP, IPv4, and IPv6
- Supports embedded scripts into the image or served by the HTTP/TFTP, and so on
- Supports stateless address autoconfiguration (SLAAC) and stateful IP autoconfiguration 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, and so on
- Unused network boot options such as FCoE, iSCSI SAN, Wireless, and so on
- Loading of unsupported NBP (such as syslinux/pxelinux) because these can 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 Nexus images which are over 250 Mbytes.
- iPXE only allows/boots Cisco signed NBI images. Other standard image format support is disabled for security reasons.

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 Nexus image. It also can 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 Nexus image. It also can 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 9

Kernel Stack

This chapter contains the following sections:

About Kernel Stack

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

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

Guidelines and Limitations

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.

  For more information about the NVE interface, see the Cisco Nexus 9000 Series NX-OS VXLAN Configuration Guide.

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, be aware of the following issues:

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

PART II

Applications

CHAPTER 10

Third-Party Applications

This chapter contains the following sections:

About Third-Party Applications

The RPMs for the Third-Party Applications are available in the repository at https://devhub.cisco.com/artifactory/open-nxos/7.0-3-I2-1/x86_64. 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** plugin gets executed. This plugin copies the RPM to a hidden location. On switch reload, the system re-installs the RPM.

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

Installing Third-Party Native RPMs/Packages

The complete workflow of package installation is as follows:

Procedure

Configure the repository on the switch to point to the Cisco repository where agents are stored.

```
bash-4.2# cat /etc/yum/repos.d/open-nxos.repo
[open-nxos]
name=open-nxos
baseurl=https://devhub.cisco.com/artifactory/open-nxos/7.0-3-I2-1/x86_64/
enabled=1
gpgcheck=0
sslverify=0
```

An example of installation of an rpm using *yum*, with full install log.

Example:

```
bash-4.2# yum install splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package splunkforwarder.x86_64 0:6.2.3-264376 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

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

 Package              Arch                 Version          Repository            Size
==========================================================================================
Installing:
splunkforwarder      x86_64               6.2.3-264376     open-nxos             13 M

Transaction Summary
==========================================================================================
Install       1 Package

Total size: 13 M
Installed size: 34 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : splunkforwarder-6.2.3-264376.x86_64
                                                                              1/1
complete

Installed:
  splunkforwarder.x86_64 0:6.2.3-264376


Complete!
bash-4.2#
```

An example of querying the switch for successful installation of the package, and verifying that its processes or services are up and running.

Example:

```
bash-4.2# yum info splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
Fretta             |  951 B     00:00 ...
groups-repo        | 1.1 kB     00:00 ...
localdb            |  951 B     00:00 ...
patching           |  951 B     00:00 ...
thirdparty         |  951 B     00:00 ...
Installed Packages
Name        : splunkforwarder
Arch        : x86_64
Version     : 6.2.3
Release     : 264376
```

```
Size        : 34 M
Repo        : installed
From repo   : open-nxos
Summary     : SplunkForwarder
License     : Commercial
Description : The platform for machine data.
```

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_n9000.rpm

bgp-1.0.0-r0.lib32_n9000.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 created in the `/etc` directory persists across reloads. For example, a third-party service script created under `/etc/init.d/` brings up the apps during 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 146.

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 package**	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 package
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, on page 94.

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_n9000 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_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.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_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.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

DevOps Configuration Management Tools

For DevOps configuration management tools, refer to the following links:

- Ansible 2.0 Release(Nexus Support), http://releases.ansible.com/ansible/
- Ansible NX-OS Sample Modules, https://github.com/jedelman8/nxos-ansible
- Puppet, https://forge.puppetlabs.com/puppetlabs/ciscopuppet
- Cisco Puppet Module(Git), https://github.com/cisco/cisco-network-puppet-module/tree/master
- Chef, https://supermarket.chef.io/cookbooks/cisco-cookbook
- Cisco Chef Cookbook(Git), https://github.com/cisco/cisco-network-chef-cookbook/tree/master

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 protototocol (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 11

Ansible

Prerequisites

Go to https://docs.ansible.com/ansible/intro_installation.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 12

Puppet Agent

This chapter includes the following sections:

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 Master (server). The Puppet Master typically runs on a separate dedicated server and serves multiple devices. The operation of the puppet agent involves periodically connecting to the Puppet Master, which in turn compiles and sends a configuration manifest to the agent. The agent reconciles this manifest with the current state of the node and updates state 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/products/faq
Puppet Labs Documentation	https://puppet.com/docs

Prerequisites

The following are prerequisites for the Puppet Agent:

- You must have a Cisco device and operating system software release that supports the installation.

    - Cisco Nexus 3500 Series switch
    - Cisco Nexus 3100 Series switch.
    - Cisco Nexus 3000 Series switch.
    - 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 Master 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 Cisco Nexus platform 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 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 Nexus NX-OS operating system and platform. This module is installed on the Puppet Master and is required for puppet agent operation on Cisco Nexus switches.

The ciscopuppet module is available on Puppet Forge.

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

ciscopuppet Module location (Puppet Forge)	https://forge.puppetlabs.com/puppetlabs/ciscopuppet
Resource Type Catalog	https://github.com/cisco/cisco-network-puppet-module/tree/master#resource-by-tech
ciscopuppet Module: Source Code Repository	https://github.com/cisco/cisco-network-puppet-module/tree/master

ciscopuppet Module: Setup & Usage	Cisco Puppet Module::README.md
Puppet Labs: Installing Modules	https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html
Puppet NX-OS Manifest Examples	https://github.com/cisco/cisco-network-puppet-module/tree/master/examples
NX-OS developer landing page.	Configuration Management Tools

CHAPTER 13

Using Chef Client with Cisco NX-OS

This chapter includes the following sections:

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 3500 Series switch
    - Cisco Nexus 3100 Series switch
    - 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 Cisco Nexus platform 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 Cisco Nexus platform (automated installation using the Chef provisioner)	cisco-cookbook::README-chef-provisioning.md

cisco-cookbook

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

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	https://supermarket.chef.io/cookbooks/cisco-cookbook

Topic	Link
Resource Type Catalog	https://github.com/cisco/cisco-network-chef-cookbook/tree/master#resource-by-tech
cisco-cookbook: Source Code Repository	https://github.com/cisco/cisco-network-chef-cookbook/tree/master
cisco-cookbook: Setup and usage	https://github.com/cisco/cisco-network-chef-cookbook/blob/master/README.md#setup
Chef Supermarket	https://supermarket.chef.io
Chef NX-OS Manifest Examples	https://github.com/cisco/cisco-network-chef-cookbook/tree/master/recipes

CHAPTER 14

Nexus Application Development - ISO

This chapter contains the following sections:

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 Cisco Nexus 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 15

Nexus Application Development - SDK

This chapter contains the following sections:

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 Cisco Nexus switch running the NX-OS Release 7.0(3)I2(1). 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: http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-I2-1/x86_64/wrlinux-5.0.1.13-eglibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.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-5.0.1.13-eglibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Enter target directory for SDK (default: /opt/windriver/wrlinux/5.0-n9000):
/path/to/sdk_install_directory
You are about to install the SDK to "/path/to/sdk_install_directory". ProceedY/n?Y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
bash$
```

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 Nexus switch, you will use the `/usr/sbin/chkconfig` instead. Edit the following edited in the spec file:

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

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

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java 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
```

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

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

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

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

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

Add the following option:

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

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

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

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java 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 16

NX-SDK

About the NX-SDK

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

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

The NX-SDK also supports Python bindings.

Note For Cisco Nexus NX-OS 7.0(3)I6(1) and earlier versions, the NX-SDK is not supported on Cisco Nexus 3000 switches.

Requirements

The NX-SDK has the following requirements:

- Docker

Install the NX-SDK

Procedure

Step 1

Note The Cisco SDK is required for applications started in VSH.

The Cisco SDK is optional for applications started in Bash.

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

a) Pull the Docker image for Ubuntu 14.04+ or Centos 6.7+ from https://hub.docker.com/r/dockercisco/nxsdk.

b) Source for a 32-bit environment:

Example:

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

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

Example:

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

What to do next

The following references to the API can be found in `$PWD/nxsdk` and includes the following:

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

Building and Packaging C++ Applications

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

Procedure

Step 1 Build your application files..

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

Example:

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

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

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

Example:

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

Step 2 (Optional) Package your application.

Auto-generate RPM package

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

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

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

Example:

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

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

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

Example:

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

Example of Auto-generate RPM package for C++ App examples/customCliApp.cpp

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

Generating rpm package...

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

+ /bin/mkdlr -p
/enxos-sdk/sysrOOts/x86_64-wrIinuxsdk-linux/usr/lib/rpm/../../../var/tmp/customCliApp-root//isan/bin
```

```
+ cp -R /NX-SDK/bin /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/..
/../../var/tmp/customCliApp-root//isan/bin
+ exit 0
Processing files: customCliApp-1.0-7.03.I6.1.x86_64
Requires: libc.so.6 libc.so.6(GLIBC 2.0) 3.0) Libc.so.6(GLIBC_2.1.3) libdl.so.2 libgcc_s.so.1
 libgcc_s.so.1(GCC_3.0) libm.so.6 libnxsdk.so libstdc++.so.6 libstdc++.so.6 (CXXAB1 1.3)
libstdc++.so.6(GLIBCXX 3.4) libstdc++.so.6(GLIBCXX_3.4.14) rt1d(GNU HASH)
Checking for unpackaged file(s):
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/check-files
/enos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../../var/tmp/customCliApp-root
Wrote:
/enxos-sdk/sysrootS/X86_64-wrlinuxsdk-linux/usr/src/rpm/SRPMS/customCliApp-1.0-7.0.3.I6.1.src-rpm

Wrote:
/enxos-sdk/sysrootS/X86_64-wrlinuxsdk-linux/usr/src/rpm/RPMS/x86_64/customCliApp-1.0-7.0.3.I6.1.x86_64.rpm
Executing($clean): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ / bin/rm -rf
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../../var/tmp/customCliApp-root

RPM package has been built
###########################################################################################

SPEC file: /NX-SDK/rpm/SPECS/customCliApp.spec
RPM file : /NX-SDK/rpm/RPMS/customCliApp-1.0-7.0.3.I6.1.x86_64.rpm
```

Manually-generate RPM Package

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

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

Example:

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

b) Enter the `$RPM_ROOT` directory.

Example:

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

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

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

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

Example:

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

d) Build your RPM package.

Example:

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

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

Installing and Running Custom Applications

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

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

To run NX-SDK apps inside the swtich (on box), you must have the Cisco SDK build environment that is installed.

Note The Cisco SDK is required to start applications in VSH: VSH requires that all applications be installed through RPMs, which requires that being built in the Cisco SDK.

The Cisco SDK is not required for Python application.

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

To install or run custom applications on the switch, use this procedure:

Before you begin

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

Procedure

Step 1 Install your application using either VSH or Bash.

To install your application using VSH, perform the following:

a) Add the RPM package to the installer.

Example:

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

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

Example:

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

c) Activate the RPM package.

Example:

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

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

Example:

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

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

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

Step 2 Start your application.

C++ applications can run from VSH or Bash.

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

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

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

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

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

Python applications can run from VSH or Bash.

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

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

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

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

- To run a Python application from Bash,

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

  Note By default, NX-SDK uses `/isan/bin/nxsdk` to run Python applications in Bash, but you can specify a different install directory if needed.

Step 3 Run **show nxsdk internal service** to verify that your application is running

Example:

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

switch(config)# show nxsdk internal service

  NXSDK total services (Max Allowed) : 2 (32)
  NXSDK Default App Path         : /isan/bin/nxsdk
  NXSDK Supported Versions       :  1.0

  Service-name              Base App        Started(PID)    Version    RPM Package
  ------------------------- --------------- ------------    ---------- --------------------
```

```
  /isan/bin/capp1           nxsdk_app2      VSH(25270)      1.0
capp1-1.0-7.0.3.I6.1.x86_64
  /isan/bin/TestApp.py      nxsdk_app3      BASH(27823)     -          -
```

Step 4 Stop you application.

You can stop your application in the following ways:

- To stop all NX-SDK applications, run **no feature nxsdk**.
- To stop a specific application in VSH, run **no nxsdk service-name** /*install directory*/*application*
- To stop a specific application in Bash, run *application* **stop-event-loop**

Step 5 Uninstall your application.

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

a) Deactivate the active RPM package.

Example:

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

b) Verify that the package is deactivated.

Example:

```
switch# show install inactive
```

c) Remove the RPM package.

Example:

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

To uninstall the RPM from the switch using Bash, run **yum remove** *app-full-path*

PART III

NX-API

CHAPTER 17

NX-API CLI

About NX-API CLI

On Cisco Nexus devices, command-line interfaces (CLIs) are run only on the device. NX-API CLI improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco Nexus CLI system on the Cisco Nexus 3000 Series devices. 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 3500 Series devices.

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 Cisco Nexus 3000 Series devices are entered using NX-API by encoding the CLIs into the body of a HTTP/HTTPs POST. The response to the request is returned in XML or JSON output format.

Note For more details about NX-API response codes, see Table of NX-API Response Codes, on page 147.

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.

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 certificate enable	Enables a certificate.
nxapi use-vrf *vrf*	Specifies the default VRF, management VRF, or named VRF. **Note** In Cisco NX-OS Release 7.0(3)I2(1) nginx listens on only one VRF.
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 146.

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)#
```

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. • For the XML output message format (XML is the default.), special characters, such as < or >, are converted to form a valid XML message (< is converted into &lt; > is converted into &gt). You can use XML SAX to parse the chunked output. **Note** When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.
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.
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-API Request Element	Description
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.

Note When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.

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 9000 Series NX-OS Security Configuration Guide*:

https://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/products-installation-and-configuration-guides-list.html

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 144.

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 146.

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.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.

CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow **show** .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive show commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Backend processing error.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

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

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

Selected examples of this feature follow.

About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard 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 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)#
```

CHAPTER 18

NX-API REST

About NX-API REST

NX-API REST

NX-API REST is available for use with the Cisco Nexus 3132Q-XL, 3172PQ-XL, and 3172TQ-XL switches [starting with Cisco NX-OS Release 7.0(3)I2(2)], and with the Cisco Nexus 3164Q and 31128PQ switches. NX-API REST is not supported for other Cisco Nexus 3000 and 3100 Series switches.

On Cisco Nexus devices, configuration is performed using command-line interfaces (CLIs) that run only on the device. NX-API REST improves the accessibility of the Nexus 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 Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see https://developer.cisco.com/site/nx-api/documents/n3k-n9k-api-ref/.

CHAPTER 19

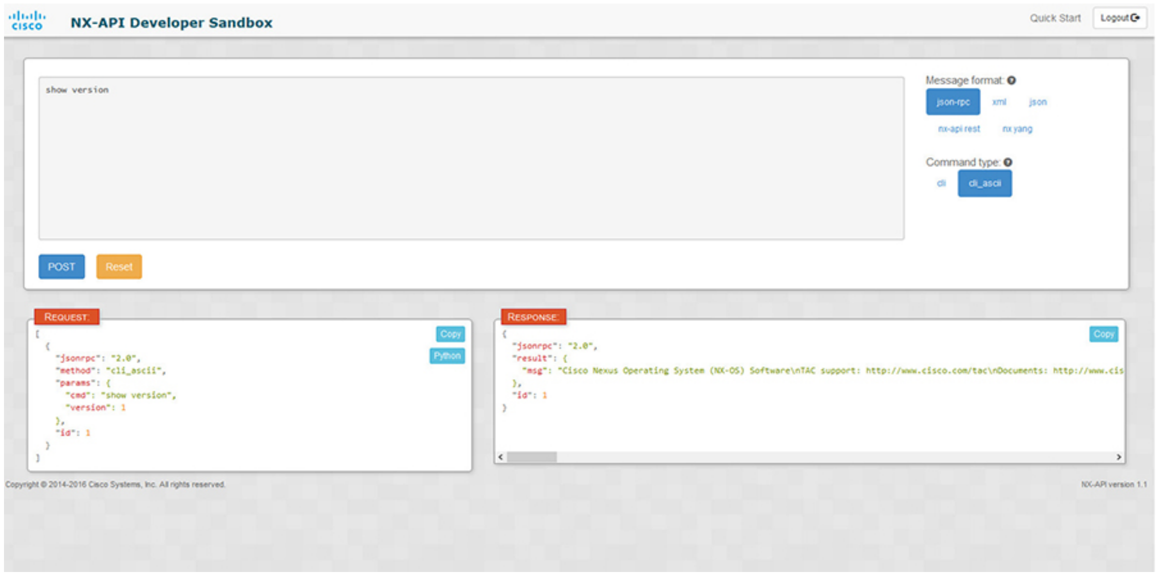
NX-API Developer Sandbox

About the NX-API Developer Sandbox

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

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

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



```
show version
```

Message format: json-rpc xml json nx-api rest nx yang

Command type: cli cli_ascii

REQUEST:

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

RESPONSE:

```
{
  "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
}
```

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

NX-API version 1.1

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 Restrictions 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 about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see https://developer.cisco.com/site/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 REST Payloads

Tip Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window.

Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI".

Only configuration commands are supported.

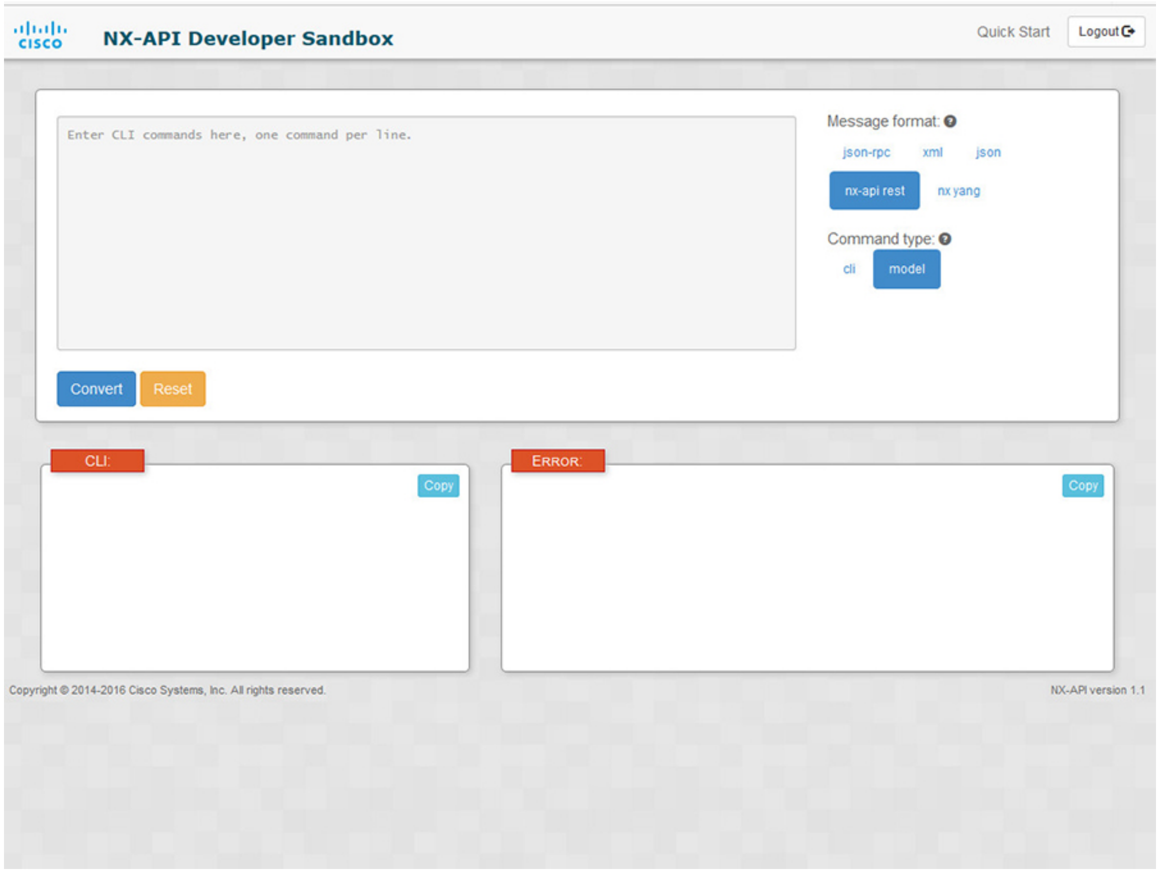
Procedure

Step 1 Configure the **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 156.

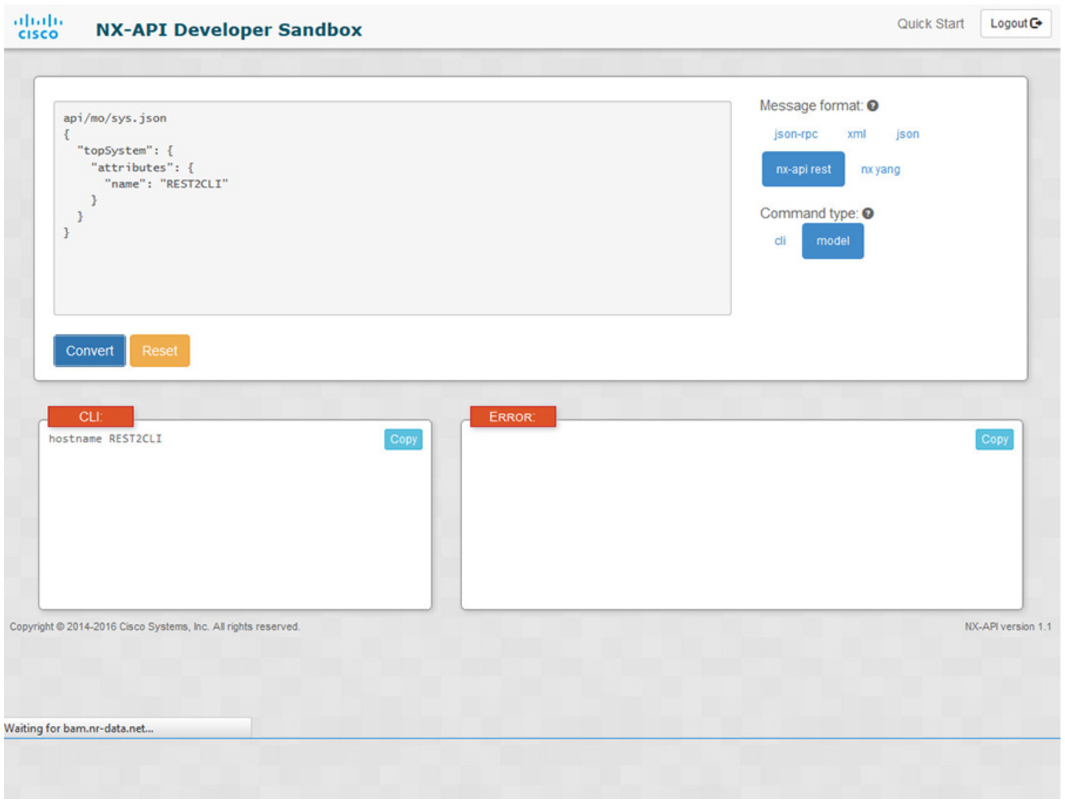
Step 2 Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.

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.



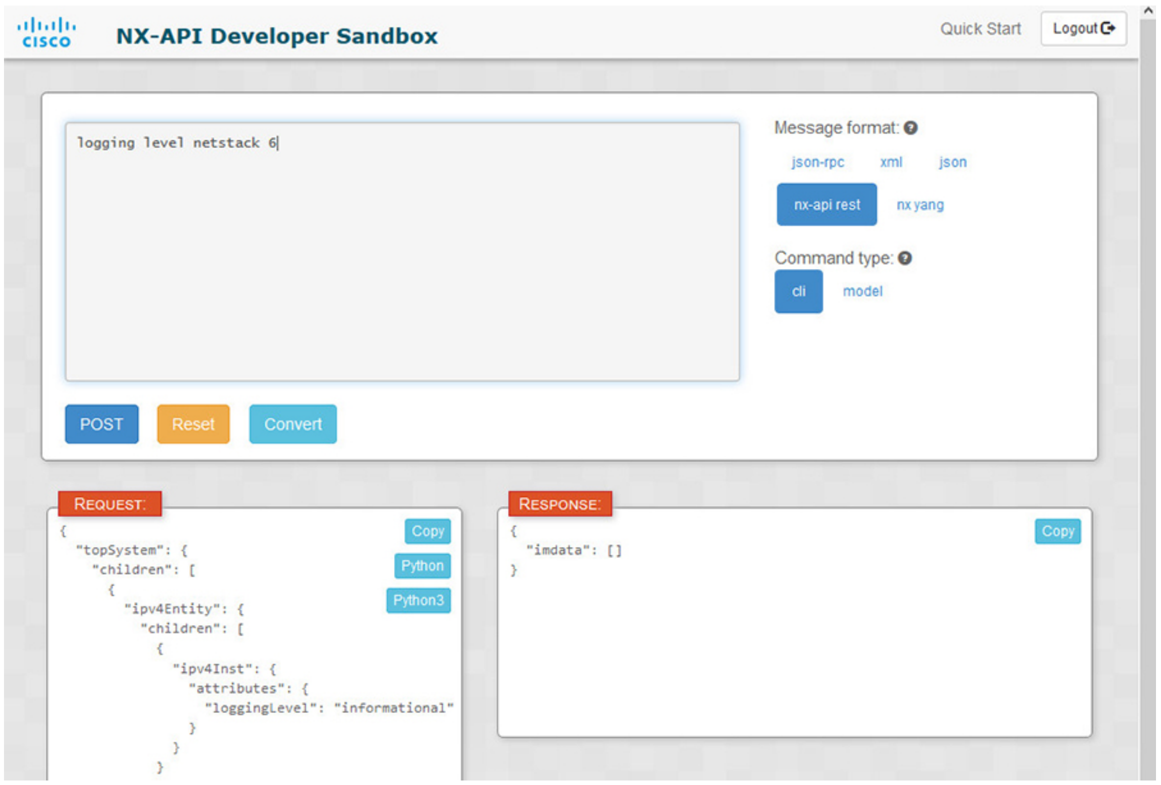
```
api/mo/sys.json
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

```
hostname REST2CLI
```

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.

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

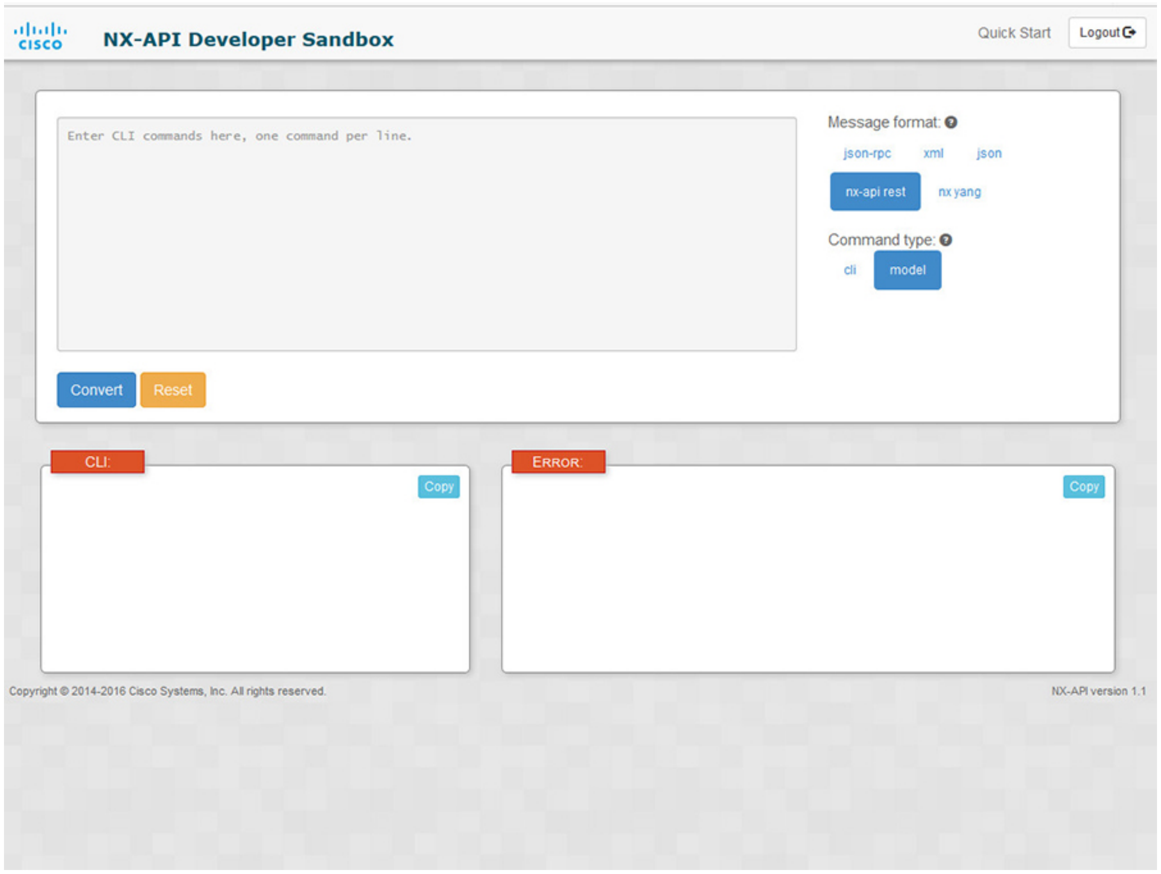
Tip Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window.

Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI".

Procedure

Step 1 Select `nx-api rest` as the **Message Format** and `model` as the **Command Type**.

Example:

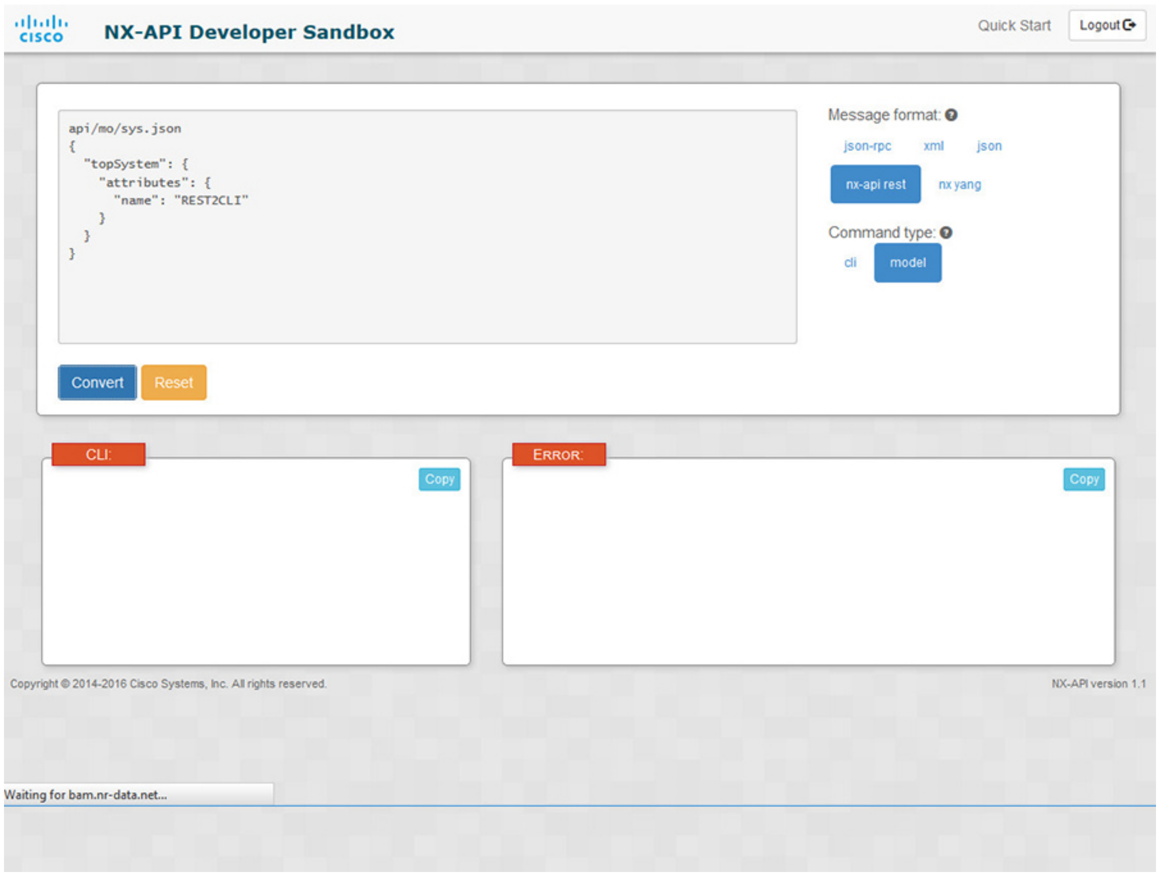


Step 2 Enter a DN and payload into the text entry box in the top pane. Then click on the **Convert** button below the top pane.

Example:

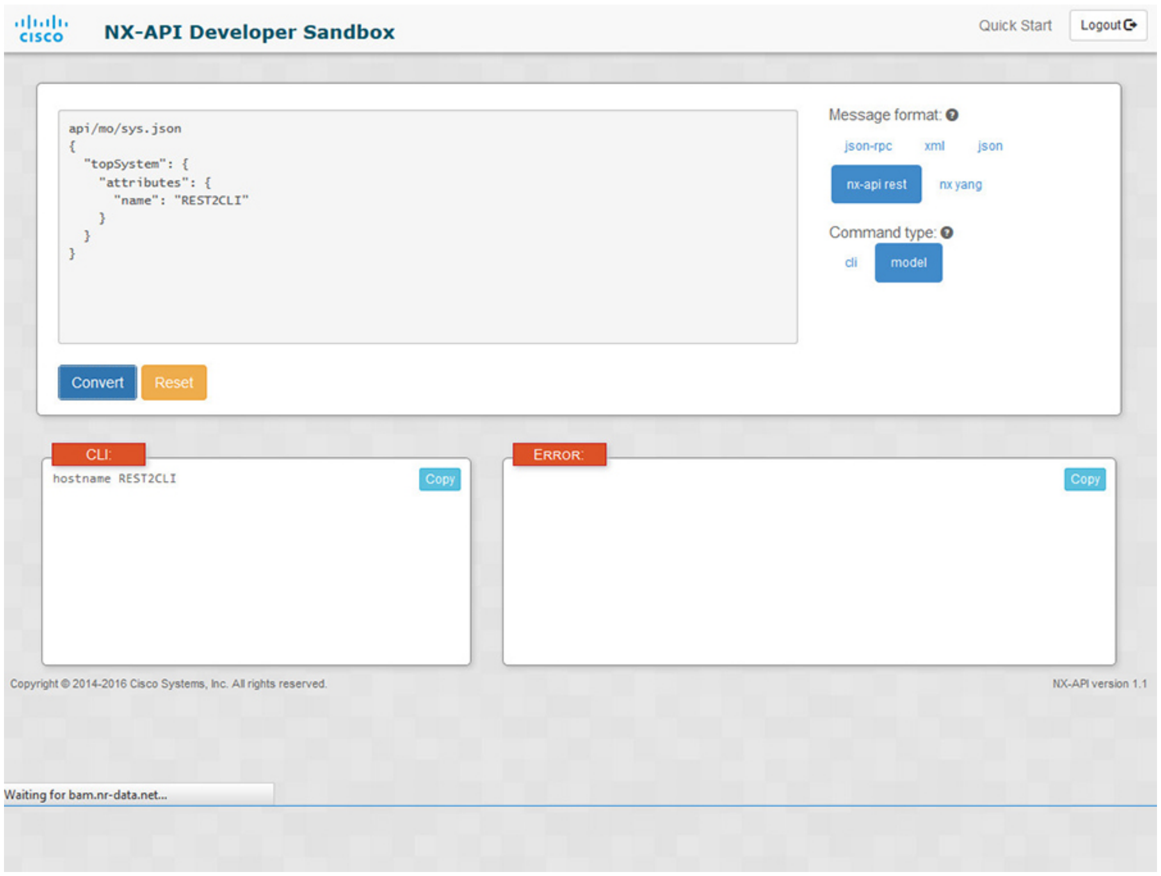
For this example, the DN is **`api/mo/sys.json`** and the NX-API REST payload is:

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



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

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



```
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...

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

Table 12: 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.`

PART IV

Model-Driven Programmability

CHAPTER 20

Infrastructure Overview

About Model-Driven Programmability

The model-driven programmability of the NX-OS device allows you to automate the configuration and control of the device.

Data Modeling

Data modeling provides a programmatic and standards-based method of writing configurations to the network device, replacing the process of manual configuration. Data models are written in a standard, industry-defined language. Although configuration using a CLI may be more human-friendly, automating the configuration using data models results in better scalability.

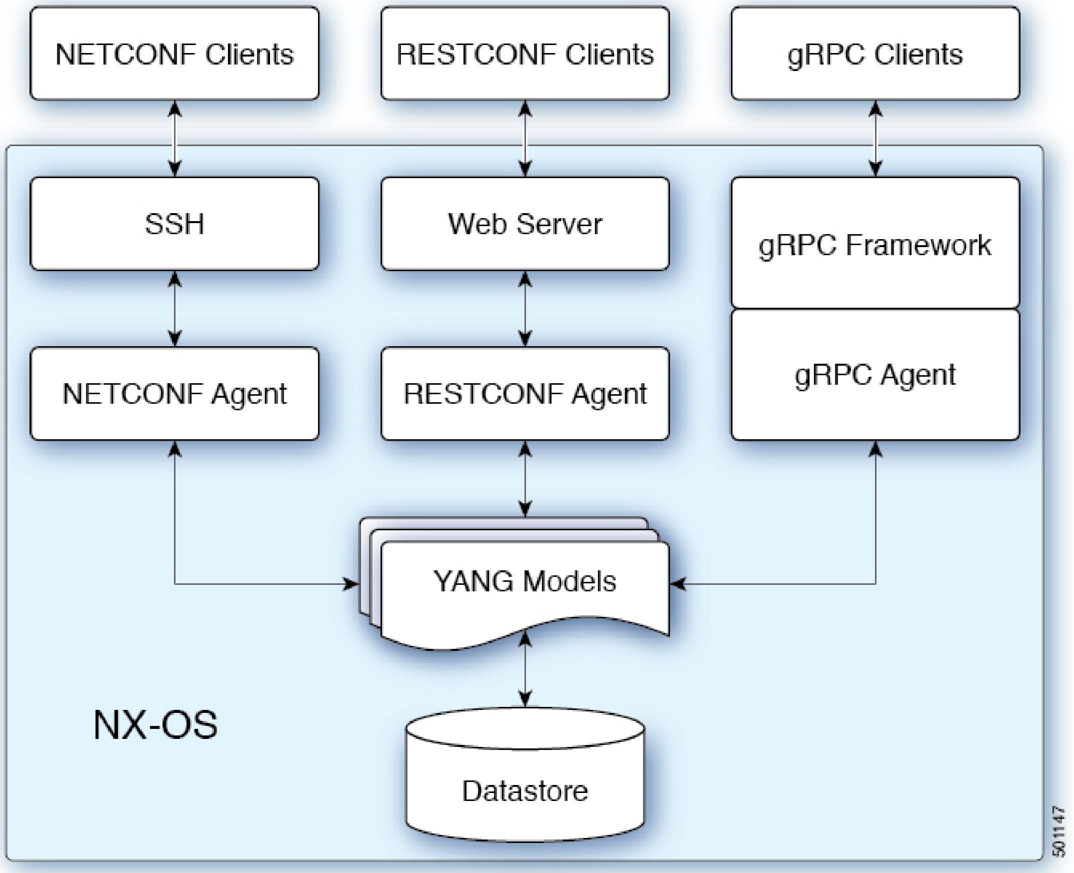
The Cisco NX-OS device supports the YANG data modeling language. YANG is a data modeling language used to describe configuration and operational data, remote procedure calls, and notifications for network devices.

Programmable Interfaces

Three standards-based programmable interfaces are supported by NX-OS for operations on the data model: NETCONF, RESTConf, and gRPC.

About the Programmable Interface Infrastructure

This section provides a brief overview of the NX-OS Programmable Interface infrastructure.



When a request is received whether via NETCONF, RESTConf, or gRPC, the request is converted into an abstract message object. That message object is distributed to the underlying model infrastructure based on the namespace in the request. Using the namespace, the appropriate model is selected and the request is passed to it for processing. The model infrastructure executes the request (read or write) on the device datastore. The results are returned to the agent of origin for response transmission back to the requesting client.

NX-OS Programmable Interface Agents

Agents provide an interface between the Device and clients. They specify the transport, the protocol, and the encoding of the communications with the Device. NX-OS Programmable Interfaces support three agents: NETCONF, RESTConf, and gRPC, each providing different interfaces for configuration management of the Device via YANG models.

Note Supported YANG models for each Cisco NX-OS release are provided at https://devhub.cisco.com/artifactory/open-nxos-agents.

Table 13: NX-OS Programmable Interface Agents

Agent	Transport	Protocol	Encoding
NETCONF	SSH		XML
RESTConf	HTTP	draft-ietf-netconf-restconf-10[1	XML or JSON
gRPC	HTTP	gRPC Protocol Spec[2]	Google Protobuf

The protocol specifications are described in the following documents:

- [1] RESTCONF Protocol draft-ietf-netconf-restconf-10 https://tools.ietf.org/html/draft-ietf-netconf-restconf-10
- [2] Cisco NX-OS gRPC Protocol Specification

Model Infrastructure

The Model Infrastructure takes requests that are received from the Agent, determines the namespace that is associated with the YANG model in the request, and selects the model component matching the namespace to process the request. When the selected model component completes request processing, the processing results are sent to the requesting Agent for transmission back to the client. The Model Infrastructure is also responsible for handling protocol initiation requests involving authentication, handshaking, and so on, as specified by the Agent protocol.

Device YANG Model

The Device Configuration is described in a YANG model that is called a Device Model. The Device Model is manifested in the Model Infrastructure as another model component with the Device namespace.

Common YANG Models

A Common Model is another kind of model component that contains within its elements, YANG Paths to the equivalent Device Model elements. These equivalent Device Model elements are used to read and write Device Model data in the Device YANG context.

Additional YANG References

Additional information about YANG can be found at the *YANG Central Wiki* http://www.yang-central.org/twiki/bin/view/Main/WebHome (M. Bjorklund, Ed.)

CHAPTER 21

Managing Components

About the Component RPM Packages

Note Beginning with Cisco Nexus NX-OS 7.0(3)I7(1), the NX-OS Programmable Interface Base Component RPM packages (agents, the Cisco native model, the majority of other required models, and infrastructure) are included in the NX-OS image. As a result, nearly all the required software is installed automatically when the image is loaded. This 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.

However, for Cisco Nexus NX-OS 7.0(3)I6(1) and earlier releases, or for the purpose of upgrading, 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. Three types of component RPM packages are needed:

- NX-OS Programmable Interface Infrastructure Components
- Common Model Components
- Agent Components

NX-OS Programmable Interface Infrastructure Components

The NX-OS Programmable Interface Infrastructure comprises two RPM packages:

- **mtx-infra** — This RPM is platform-independent.
- **mtx-device-model** — This RPM is *platform-dependent* and must be selected to match the installed NX-OS image at the time of Cisco Artifactory download.

Common Model Components

Common Model component RPMs provides support for Openconfig and IETF defined models. In order to enable support for one or more desired Common Models, the associated Common Model component RPMs must be downloaded and installed. As with the mtx-device-model RPM, Common Model components are also *platform-dependent* and must be selected to match the installed NX-OS image at the time of Cisco Artifactory download.

Following is the list of supported RPMs:

- `mtx-openconfig-bgp`
- `mtx-openconfig-bgp-multiprotocol`
- `mtx-openconfig-if-ip`
- `mtx-openconfig-interfaces`
- `mtx-openconfig-local-routing`
- `mtx-openconfig-routing-policy`
- `mtx-openconfig-vlan`

Agent Components

Three agent packages are available: NETCONF, RESTConf and gRPC. At least one agent must be installed in order to have access to the modeled NX-OS interface.

Note Beginning with Cisco Nexus NX-OS 7.0(3)I7(3), you can enable the agents using the **feature netconf** and **feature restconf** commmands. However, these commands are not included in prior releases, so for releases Cisco Nexus NX-OS 7.0(3)I7(1) to 7.0(3)I7(3), you must enter the bash shell using the instructions in the "Opening the Bash Shell on the Device" section in Preparing For Installation, on page 174, then run the **netconfctl start** and **restconfctl start** commands.

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 **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 in progress, the RPM installation may fail.

  ```
  Switch# show system redundancy status
  ```

If the line cards have failed to come up, enter the **`createrepo /rpms`** command in the bash shell.

```
bash-4.2# createrepo /rpms
```

Copying Files to the Device

You can use SCP to copy files to the device, using a command in this form.

copy scp://*username@source_ip/path_to_agent_rpm* **bootflash: vrf management**

Example:

```
Switch# copy scp://jdoe@192.0.20.123//myrpms/mtx-infra.1.0.0.r082616.x86_64.rpm bootflash:
 vrf management
```

Displaying Installed NX-OS Programmable Interface RPMs

To show all installed NXOS Programmable Interface RPMs, issue the following command on the device:

```
bash-4.2# yum list installed | grep mtx
```

Downloading Components from the Cisco Artifactory

The NX-OS Programmable Interface Component RPMs can be downloaded from the Cisco Artifactory at the following URL:

https://devhub.cisco.com/artifactory/open-nxos-agents

The NX-OS Programmable Interface Component RPMs adhere to the following naming convention:

mtx-*name***.***ma.mi.ma***.r***YYYYYY***.x86_64.rpm**

or

mtx-*name-XXXX***.***ma.mi.ma***.r***YYYYYY***.x86_64.rpm**

where

- *name* — MTX component name (such as infra, device-model, openconfig-interface, netconf, etc)
- *ma.mi.ma* — RPM Version Number (major.minor.maintenance)

- *XXXX* — NX-OS Image ID
- **r***YYYYYY* — RPM build ID

Select and download the desired NX-OS Programmable Interface Component RPM packages to a local server for installation on the device as described in the following sections.

Installing RPM Packages

Installing the Programmable Interface Infrastructure RPM Packages

Before you begin

- From the Cisco Artifactory, download the following packages:
  - mtx-infra.ma.mi.ma.rYYYYYY.x86_64.rpm (Infrastructure)
  - mtx-device-XXXX.ma.mi.ma.rYYYYYY.x86_64.rpm RPM (Device Model)
- Using the CLI commands in Verify Device Readiness, on page 175, 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.1.0.0.r082616.x86_64.rpm bootflash:
 vrf management
Switch# copy scp://jdoe@192.0.20.123//myrpms/mtx-device.1.0.0.r082616.x86_64.rpm bootflash:
 vrf management
```

Step 2 From the bash shell, install the MTX Infra RPM.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-infra.1.0.0.r082616.x86_64.rpm
```

Step 3 From the bash shell, install the MTX Device Model RPM.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-device.1.0.0.r082616.x86_64.rpm
```

Step 4 From the bash shell, verify the installation.

Example:

```
bash-4.2# yum list installed | grep mtx
```

What to do next

Install one or more Common Model RPM Packages.

Installing Common Model RPM Packages

Before you begin

- Install the Programmable Interface Infrastructure RPM packages.
- Download the Common Model RPM packages from the Cisco Artifactory.

Procedure

Step 1 Copy the downloaded Common Model RPMs to the device.

Example:

```
bash-4.2# copy
scp://jdoe@192.0.20.123//myrpms/mtx-openconfig_interfaces.1.0.0.r082616.x86_64.rpm bootflash:
 vrf management
```

Step 2 From the bash shell, install each Common Model RPM.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-openconfig_interfaces.1.0.0.r082616.x86_64.rpm
```

Step 3 From the bash shell, verify the installation.

Example:

```
bash-4.2# yum list installed | grep mtx
```

What to do next

Install one or more NX-OS Programmable Interface Agent RPM Packages.

Installing Agent RPM Packages

Before you begin

- Install the Programmable Interface Infrastructure RPM packages.
- Install one or more Common Model RPM packages.
- Download the Agent RPM packages from the Cisco Artifactory.

Procedure

Step 1 Copy the downloaded Agent RPMs to the device.

Example:

```
bash-4.2# copy scp://jdoe@192.0.20.123//myrpms/myrpms/mtx-netconf.1.0.0.r082616.x86_64.rpm
 bootflash: vrf management
```

Step 2 From the bash shell, install the Agent RPM.

Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-netconf.1.0.0.r082616.x86_64.rpm
```

Step 3 From the bash shell, verify the installation.

Example:

```
bash-4.2# yum list installed | grep mtx
```

What to do next

Configure the Agent.

CHAPTER 22

NETCONF Agent

About the NETCONF Agent

The Cisco NX-OS NETCONF Agent is a client-facing interface that provides secure transport for the client requests and server responses in the form of a YANG model, encoded in XML.

The NETCONF Agent supports a candidate configuration feature. The Candidate configuration datastore temporarily holds candidate configuration and any changes you make without changing the running configuration. You can then choose when to update the configuration of the device with the candidate configuration when you commit and confirm the candidate configuration.

If you do not confirm the changes, exit from a nonpersistent NETCONF client session, or choose to cancel the commit after you commit the change, a system timer then times out and rolls back the changes if you do not confirm the changes.

If you initiate a confirmed-commit operation with a persistent token, the NETCONF client session becomes a persistent process. In a persistent process, exiting the NETCONF client session will not call an automatic roll-back and the changes cannot be rolled back without the matching persistent token.

Cisco NX-OS NETCONF supports the following configuration capabilities:

- Writable-Running Capability

  ```
  urn:ietf:params:netconf:capability:writable-running:1.0
  ```

- Rollback-on-error Capability

  ```
  urn:ietf:params:netconf:capability:rollback-on-error:1.0
  ```

- Candidate Configuration Capability

  ```
  urn:ietf:params:netconf:capability:candidate:1.0
  ```

- Validation Capability

  ```
  urn:ietf:params:netconf:capability:validate:1.1
  ```

- Confirmed Commit Capability

```
    urn:ietf:params:netconf:capability:confirmed-commit:1.1
```

When a new session starts, the NETCONF Agent sends out a <hello> message advertising its capabilities. The following example shows a NETCONF agent sending a <hello> message to the client:

```
<?xml version="1.0" encoding="UTF-8"?>
<hello>

 <capabilities>

  <capability>urn:ietf:params:netconf:base:1.0</capability>
  <capability>urn:ietf:params:netconf:base:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>

<capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2017-04-06&amp;module=cisco-nx-os-device&amp;deviations=cisco-nx-os-device-deviations</capability>

 </capabilities>

 <session-id>1438752697</session-id>
</hello>
```

The Cisco NX-OS NETCONF Agent supports the following NETCONF Protocol operations:

- get
- get-config
- edit-config
- close-session
- kill-session

Candidate configuration supports the following NETCONF Protocol operations.

- Operations for the candidate configuration as <source> or <target>.
    - get-config
    - edit-config
    - copy-config
    - lock
    - unlock
    - validate
- Operations for the candidate configuration that do not require explicitly specifying the candidate configuration as <source> or <target>.
    - commit
    - cancel-commit
    - discard-changes

Note The delete-config operation is not allowed.

Guidelines and Limitations

The NETCONF Agent has the following guideline and limitation:

- All Cisco Nexus 3100-V, 3100-XL, and the 3200 platform switches support NETCONF.

Configuring the NETCONF Agent

The NETCONF Agent supports the following optional configuration parameters under the `[netconf]` section in the configuration file (`/etc/mtx.conf`).

Parameter	Description
idle_timeout	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The default value is 5 minutes. A value of 0 disables timeout.
limit	(Optional) Specifies the number of maximum simultaneous client sessions. The default value is 5 sessions. The range is 1 to 10.

The following is an example of the `[netconf]` section in the configuration file:

```
[netconf]
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

For the modified configuration file to take effect, you must restart the NETCONF Agent using the CLI command [**no**] **feature netconf** to disable and re-enable.

Using the NETCONF Agent

General Commands

The NETCONF Agent is enabled or disabled by the command [**no**] **feature netconf**.

Initializing the Candidate Configuration Datastore

The candidate configuration can only be initialized with the contents of the running configuration. To Initialize the candiate configuring datastore, send a Copy-Config request using SSH, with candidate as the target and running as the source.

Performing Read and Write on the Candidate Configuration

To read from the candidate configuration, send a Get-Config request with SSH, using candidate as the source.

To write to the contents of the candidate configuration, send an Edit-Config request with SSH, using candidate as the target.

NETCONF Candidate Configuration Workflow

The candidate configuration workflow is as follows:

- Edit the candidate configuration file.
- Validate the candidate configuration.
- Commit the changes to the running configuration.

Example: An SSH Session

This example shows initiating a session using the SSH client and sending an Edit-Config and Get request using the SSH Client.

```
client-host % ssh -s admin@172.19.193.152 -p 830 netconf
client-host % ssh -s admin@172.19.193.152 -p 830 netconf
User Access Verification
Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello>
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>

<capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2017-04-06&amp;module=cisco-nx-os-device&amp;deviations=cisco-nx-os-device-deviations</capability>

    </capabilities>
    <session-id>1912037714</session-id>
</hello>
]]>]]><hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
#794
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
         <target>
```

```
                <running/>
            </target>
            <config>
                <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                        <bgp-items>
                            <inst-items>
                                <dom-items>
                                    <Dom-list>
                                        <name>default</name>
                                           <rtrId>2.2.2.2</rtrId>
                                    </Dom-list>
                                </dom-items>
                            </inst-items>
                        </bgp-items>
                </System>
            </config>
        </edit-config>
</rpc>
##

#190
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>

##

#511
<rpc message-id="109"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
    <source>
        <running/>
    </source>
    <filter type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list/>
                        </dom-items>
                    </inst-items>
                </bgp-items>
        </System>
    </filter>
</get-config>
</rpc>
##

#996
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="109"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
        <System>
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            <always>disabled</always>
                            <bestPathIntvl>300</bestPathIntvl>
                            <holdIntvl>180</holdIntvl>
                            <kaIntvl>60</kaIntvl>
                            <maxAsLimit>0</maxAsLimit>
                            <pfxPeerTimeout>30</pfxPeerTimeout>
                            <pfxPeerWaitTime>90</pfxPeerWaitTime>
                            <reConnIntvl>60</reConnIntvl>
                            <rtrId>2.2.2.2</rtrId>
                        </Dom-list>
                    </dom-items>
                </inst-items>
            </bgp-items>
        </System>
    </data>
</rpc-reply>

##
```

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

- create
- merge
- delete

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

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

Error Messages

If a request results in an error, the response payload includes the error.

Errors Defined by Cisco

The following are the errors defined by Cisco.

Error defined by Cisco	Description
unknown-error-cond	Unknown error encountered.

n-y-i	The requested operation is not supported. (not-yet-implemented).
namespace-not-found	Error in request payload.
namespace-already-exists	Error in request payload.
object-not-found	Error in request payload.
object-not-container	Error in request payload.
object-not-property	Error in request payload.
no-property-in-object	Error in request payload.
invalid-dn	Internal error.
invalid-arg	Internal error.
already-exists	Error in request payload.
container-not-found	Error in request payload
container-already-exists	Error in request payload.
property-not-found	Error in request payload.
property-already-exists	Error in request payload.
malformed	Error in request payload.
alloc-failed	Internal error.
sigint	Internal error.
not-initialized	Internal error.
inappropriate	Internal error.

The following is an example of a NETCONF error response payload that reports an invalid IP address value:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="320" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>Protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>Error</error-severity>
    <error-message xml:lang="en">Property Merge (set property) Failed: operation-failed
value=500.500.500.500</error-message>
    <error-path>/config/System/bgp-items/inst-items/dom-items/Dom-list/rtrId</error-path>
  </rpc-error>
</rpc-reply>
```

Troubleshooting the NETCONF Agent

Troubleshooting Connectivity

- From a client system, ping the management port of the switch to verify that the switch is reachable.
- In the bash shell of the switch, execute the **service netconf status** command to check the agent status.

- Check whether the RSA host key for SSH is outdated. If this is the case, remove the RSA host key entry of the switch from the `~/.ssh/known_hosts` file on the client host. This example shows the message received when the host key is outdated:

```
lient-host % ssh -s admin@192.0.20.111 -p 830 netconf
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@ @ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-
the-middle attack)! It is also possible that the RSA host
key has just been changed. The fingerprint for the RSA key
sent by the remote host is
82:3d:49:5c:1b:08:4c:8e:19:94:a8:1f:32:8d:1e:dd. Please
contact your system administrator. Add correct host key
in /users/myuser/.ssh/known_hosts to get rid of this
message. Offending key in
/users/myuser/.ssh/known_hosts:304 Password
authentication is disabled to avoid man-in-the-middle
attacks. Keyboard-interactive authentication is disabled
to avoid man-in-the-middle attacks. User Access
Verification Permission denied
(publickey,password,keyboard-interactive).
client-host %
```

CHAPTER 23

Converting CLI Commands to Network Configuration Format

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 14: 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 24

RESTConf Agent

About the RESTCONF Agent

Cisco NX-OS RESTCONF is an HTTP -based protocol for configuring data that are defined in YANG version 1, using datastores defined in NETCONF.

NETCONF defines configuration datastores and a set of Create, Retrieve, Update, and Delete (CRUD) operations that can be used to access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and event notifications.

Cisco NX-OS RESTCONF uses HTTP operations to provide CRUD operations on a conceptual datastore containing YANG-defined data. This data is compatible with a server which implements NETCONF datastores.

The RESTCONF protocol supports both XML and JSON payload encodings. User authentication is done through the HTTP Basic Authentication.

The following table shows the Protocol operations that the Cisco NX-OS RESTCONF Agent supports:

RESTCONF	NETCONF Equivalent
OPTIONS	NETCONF: none
HEAD	NETCONF: none
GET	NETCONF: <get-config>, <get>
POST	NETCONF: <edit-config> (operation="create")
PUT	NETCONF: <edit-config> (operation="create/replace")
PATCH	NETCONF: <edit-config> (operation="merge")
DELETE	NETCONF: <edit-config> (operation="delete")

Guidelines and Limitations

The RESTCONF Agent has the following guideline and limitation:

- All Cisco Nexus 3100-V, 3100-XL, and the 3200 platform switches support RESTCONF.

Configuring the RESTConf Agent

The RESTConf Agent does not require any configuration in the configuration file (**/etc/mtx.conf**).

Using the RESTCONF Agent

General Commands

- Configure the following commands to enable HTTP or HTTPS access:
  - **feature nxapi**
  - **nxapi http port 80**
  - **nxapi https port 443**

General Control Commands

You can enable or disable the RESTCONF Agent [**no**] **feature restconf** command.

Viewing the Agent Status

To view the status of the RESTCONF agent, use the **show feature** command and include the expression restconf.

```
switch-1# show feature | grep restconf
restconf               1          enabled
switch-1#
```

Sending a POST Request to the Server Using Curl

```
client-host % curl -X POST -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Content-Type:
application/yang.data+xml" -d '<always>enabled</always><rtrId>2.2.2.2</rtrId>'
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list=default"
 -i

HTTP/1.1 201 Created
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:25:31 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500853169574134
Status: 201 Created
```

```
Location: /System/bgp-items/inst-items/dom-items/Dom-list=default/always/rtrId/
```

Sending a GET Request to the Server Using Curl

```
client-host % curl -X GET -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Accept:
application/yang.data+xml"
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list?content=config"
 -i

HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:26:03 GMT
Content-Type: application/yang.data+xml
Content-Length: 395
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500856185650327
Status: 200 OK

    <Dom-list>
        <name>default</name>
        <always>enabled</always>
        <bestPathIntvl>300</bestPathIntvl>
        <holdIntvl>180</holdIntvl>
        <kaIntvl>60</kaIntvl>
        <maxAsLimit>0</maxAsLimit>
        <pfxPeerTimeout>30</pfxPeerTimeout>
        <pfxPeerWaitTime>90</pfxPeerWaitTime>
        <reConnIntvl>60</reConnIntvl>
        <rtrId>2.2.2.2</rtrId>
    </Dom-list>
client-host %
```

Troubleshooting the RESTCONF Agent

Troubleshooting Connectivity

- Enable the web server by issuing the **feature nxapi** command.
- Ensure that the **nxapi http port 80** command is configured to open up the port for HTTP
- Ensure that the **nxapi https port 443** command is configured to open up the port for HTTPS.
- Ping the management port of the switch to verify that the switch is reachable.

CHAPTER 25

gRPC Agent

About the gRPC Agent

The Cisco NX-OS gRPC protocol defines a mechanism through which a network device can be managed and its configuration data can be retrieved and installed. The protocol exposes a complete and formal Application Programming Interface (API) that clients can use to manage device configurations.

The Cisco NX-OS gRPC protocol uses a remote procedure call (RPC) paradigm where an external client manipulates device configurations utilizing Google Protocol Buffer (GPB)-defined API calls along with their service- specific arguments. These GPB-defined APIs transparently cause an RPC call to the device that return replies in the same GPB-defined API context.

The gRPC Agent provides a secure transport through TLS and user authentication and authorization through AAA.

The functional objective of the Cisco NX-OS gRPC protocol is to mirror that provided by NETCONF, particularly in terms of both stateless and stateful configuration manipulation for maximum operational flexibility.

The Cisco NX-OS gRPC Agent supports the following protocol operations:

- Get
- GetConfig
- GetOper
- EditConfig
- StartSession
- CloseSession
- KillSession

The gRPC Agent supports two types of operations:

- **Stateless operations** are performed entirely within a single message without creating a session.
- **Stateful operations** are performed using multiple messages. The following is the sequence of operations that are performed:
    1. Start the session. This action acquires a unique session ID.
    2. Perform session tasks using the session ID.
    3. Close the session. This action invalidates the session ID.

The following are the supported operations. See the Appendix for their RPC definitions in the **.proto** file that is exported by the gRPC Agent.

Operation	Description
StartSession	Starts a new session between the client and server and acquires a unique session ID.
EditConfig	Writes the specified YANG data subset to the target datastore.
GetConfig	Retrieves the specified YANG configuration data subset from the source datastore.
GetOper	Retrieves the specified YANG operational data from the source datastore.
Get	Retrieves the specified YANG configuration and operational data from the source datastore.
KillSession	Forces the termination of a session.
CloseSession	Requests graceful termination of a session.

GetConfig, GetOper, and Get are stateless operations so don't require a session ID.

EditConfig can be either stateless or stateful. For a stateless operation, specify the SessionID as 0. For a stateful operation, a valid (nonzero) SessionID is required.

The gRPC Agent supports timeout for the sessions. The idle timeout for sessions can be configured on the device, after which idle sessions are closed and deleted.

Guidelines and Limitations

The gRPC Agent has the following guideline and limitation:

- All Cisco Nexus 3100-V, 3100-XL, and the 3200 platform switches support gRPC.

Configuring the gRPC Agent for Cisco NX-OS Release 9.3(2) and Earlier

The gRPC Agent supports the following configuration parameters under the **[grpc]** section in the configuration file (**/etc/mtx.conf**).

Parameter	Description
idle_timeout	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The default timeout is 5 minutes. A value of 0 disables timeout.
limit	(Optional) Specifies the number of maximum simultaneous client sessions. The default limit is 5 sessions. The range is from 1 through 50.
lport	(Optional) Specifies the port number on which the gRPC Agent listens. The default port is 50051.
key	Specifies the key file location for TLS authentication. The default location is **/opt/mtx/etc/grpc.key**.
cert	Specifies the certificate file location for TLS authentication. The default location is **/opt/mtx/etc/grpc.pem**.
security	Specifies the type of secure connection. Valid choices are: • **TLS** for TLS • **NONE** for an insecure connection

Using the gRPC Agent

General Commands

You can enable or disable the gRPC Agent by issuing the [**no**] **feature grpc** command.

Example: A Basic Yang Path in JSON Format

```
client-host % cat payload.json
{
  "namespace": "http://cisco.com/ns/yang/cisco-nx-os-device",
  "System": {
    "bgp-items": {
      "inst-items": {
        "dom-items": {
          "Dom-list": {
            "name": "default",
            "rtrId": "7.7.7.7",
            "holdIntvl": "100"
          }
        }
      }
    }
  }
}
```

Note The JSON structure has been pretty-formatted here for readability.

Sending an EditConfig Request to the Server

```
client-host % ./grpc_client -username=admin -password=cisco -operation=EditConfig
-e_oper=Merge -def_op=Merge -err_op=stop-on-error -infile=payload.json -reqid=1
-source=running -tls=true -serverAdd=192.0.20.123 -lport=50051

######################################################
Starting the client service
######################################################
TLS set true for client requests1ems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending EditConfig request to the server
sessionid is
0
reqid:1
{"rpc-reply":{"ok":""}}
```

Sending a GetConfig Request to the Server

```
client-host % ./grpc_client -username=admin -password=cisco -operation=GetConfig
-infile=payload.json -reqid=1 -source=running -tls=true -serverAdd=192.0.20.123 -lport=50051

######################################################
Starting the client service
######################################################
TLS set true for client requests1ems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending GetConfig request to the server
in get config
```

```
Got the response from the server
#########################################
Yang Json is:
#########################################
{"rpc-reply":{"data":{"System":{"bgp-items":{"inst-items":{"dom-items":{"Dom-list":{"name":"default","rtrId":"7.7.7.7","holdIntvl":"100"}}}}}}}}
#########################################
client-host %
```

Troubleshooting the gRPC Agent

Troubleshooting Connectivity

- From a client system, verify that the agent is listening on the port. For example:

```
client-host % nc -z 192.0.20.222 50051
Connection to 192.0.20.222 50051 port [tcp/*] succeeded!
client-host % echo $?
0
client-host %
```

- In the NX-OS, check the gRPC agent status by issuing **show feature | grep grpc**.

gRPC Protobuf File

The gRPC Agent exports the supported operations and data structures in the proto definition file at `/opt/mtx/etc/nxos_grpc.proto`. The file is included in the gRPC Agent RPM. The following shows the definitions:

```
// Copyright 2016, Cisco Systems Inc.
// All rights reserved.

syntax = "proto3";

package NXOSExtensibleManagabilityService;

// Service provided by Cisco NX-OS gRPC Agent
service gRPCConfigOper {

    // Retrieves the specified YANG configuration data subset from the
    // source datastore
    rpc GetConfig(GetConfigArgs) returns(stream GetConfigReply) {};

    // Retrieves the specified YANG operational data from the source datastore
    rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

    // Retrieves the specified YANG configuration and operational data
    // subset from the source datastore
    rpc Get(GetArgs) returns(stream GetReply){};

    // Writes the specified YANG data subset to the target datastore
    rpc EditConfig(EditConfigArgs) returns(EditConfigReply) {};

    // Starts a new session between the client and server and acquires a
    // unique session ID
    rpc StartSession(SessionArgs) returns(SessionReply) {};

    // Requests graceful termination of a session
    rpc CloseSession(CloseSessionArgs) returns (CloseSessionReply) {};

    // Forces the termination of a session
    rpc KillSession(KillArgs) returns(KillReply) {};

// Unsupported; reserved for future
    rpc DeleteConfig(DeleteConfigArgs) returns(DeleteConfigReply) {};

    // Unsupported; reserved for future
    rpc CopyConfig(CopyConfigArgs) returns(CopyConfigReply) {};

    // Unsupported; reserved for future
    rpc Lock(LockArgs) returns(LockReply) {};

    // Unsupported; reserved for future
    rpc UnLock(UnLockArgs) returns(UnLockReply) {};

    // Unsupported; reserved for future
    rpc Commit(CommitArgs) returns(CommitReply) {};

    // Unsupported; reserved for future
    rpc Validate(ValidateArgs) returns(ValidateReply) {};

    // Unsupported; reserved for future
    rpc Abort(AbortArgs) returns(AbortReply) {};

}

message GetConfigArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;
}

message GetConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message GetOperArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;
```

```
}

message GetOperReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message GetArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath=1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;
}

message GetReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message EditConfigArgs
{
    // JSON-encoded YANG data to be edited
    string YangPath = 1;

    // Specifies the operation to perform on teh configuration datastore with
    // the YangPath data.  Possible values are:
    //   create
    //   merge
    //   replace
    //   delete
    //   remove
    // If not specified, default value is "merge".
    string Operation = 2;

    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 3;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 4;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 5;

    // Specifies the default operation on the given object while traversing
    // the configuration tree.
    // The following operations are possible:
```

```
    //   merge:     merges the configuration data with the target datastore;
    //              this is the default.
    //   replace:   replaces the configuration data with the target datastore.
    //   none:      target datastore is unaffected during the traversal until
    //              the specified object is reached.
    string DefOp = 6;

    // Specifies the action to be performed in the event of an error during
    // configuration.  Possible values are:
    //   stop
    //   roll-back
    //   continue
    // Default is "roll-back".
    string ErrorOp = 7;
}

message EditConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If EditConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message DeleteConfigArgs
{
    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message DeleteConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If DeleteConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CopyConfigArgs
{
    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 4;
}

message CopyConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CopyConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message LockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID=2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message LockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Lock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message UnLockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message UnLockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If UnLock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message SessionArgs
{
    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 1;
}

message SessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;
    int64 SessionID = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CloseSessionArgs
{
    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 1;

    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 2;
}

message CloseSessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CloseSession is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message KillArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    int64 SessionIDToKill = 2;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 3;
}

message KillReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Kill is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
```

```
}

message ValidateArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;
}

message ValidateReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Validate is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CommitArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;
}

message CommitReply
{
    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 1;

    // If Commit is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message AbortArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 2;
}

message AbortReply
{
    // (Optional) Specifies the request ID.  Default value is 0.
    int64 ReqID = 1;

    // If Abort is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
```

```
}
```

CHAPTER 26

Dynamic Logger

Prerequisites

Before using dynamic logging, confirm that the following are on your switch:

- The `libmtxlogmgr*.so` library is installed `/opt/mtx/lib/`. The `libmtxlogmgr*.so` library is part of the `mtx_infra` RPM.
- The `mtx.conf` file that is located in `/etc/` contains:

```
[mtxlogger]
config=/opt/mtx/conf/mtxlogger.cfg
```

- The `mtxlogger.cfg` file is in `/opt/mtx/conf/`.

Reference

The configuration file has the following structure:

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false"></leaf>
      <leaf name="allActive" type="boolean" default="false"></leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$"></leaf>
     <container name="componentID">
            <leaf name="enabled" type="boolean" default="true"></leaf>
     </container>
     <container name="dateTime">
            <leaf name="enabled" type="boolean" default="true"></leaf>
            <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
     </container>
     <container name="fcn">
            <leaf name="enabled" type="boolean" default="true"></leaf>
            <leaf name="format" type="string"
default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></leaf>
     </container>
```

```
      </container>
      <container name="dest">
        <container name="console">
          <leaf name="enabled" type="boolean" default="false"></leaf>
        </container>
        <container name="file">
          <leaf name="enabled" type="boolean" default="false"></leaf>
          <leaf name="name" type="string" default="mtx-internal.log"></leaf>
          <leaf name="location" type="string" default="./mtxlogs"></leaf>
    <leaf name="mbytes-rollover" type="uint32" default="10"></leaf>
    <leaf name="hours-rollover" type="uint32" default="24"></leaf>
    <leaf name="startup-rollover" type="boolean" default="false"></leaf>
          <leaf name="max-rollover-files" type="uint32" default="10"></leaf>
        </container>
      </container>
      <list name="logitems" key="id">
        <listitem>
         <leaf name="id" type="string"></leaf>
   <leaf name="active" type="boolean" default="true"></leaf>
        </listitem>
      </list>
    </container>
  </container>
</config>
```

The **<list>** tag defines the log filters by **<componentID>**.

The following table describes some of the containers and their leaves.

Table 15: Container and Leaf Descriptions

Container	Container Description	Contained Containers	Contained Leaf and Description
logging	Contains all logging data types	format dest file **Note** Also contains list tag "logitems"	enabled: Boolean that determines whether logging is on or off. Default off. allActive: Boolean that activates all defined logging items for logging. Default off

Container	Container Description	Contained Containers	Contained Leaf and Description
format	Contains the log message format information	componentID dateTime type fcn	content: String listing data types included in log messages. Includes: • $DATETIME$: Include date or time in log message • $COMPONENTID$: Include component name in log message. • $TYPE$: Includes message type ("", INFO, WARNING, ERROR) • $SRCFILE$: Includes name of source file. • $SRCLINE$: Include line number of source file • $FCNINFO$ Include class::function name from the source file. • MSG: Include actual log message text.
componentID	Name of logged component.	NA	enabled: Boolean that determines if the log message includes the component ID. Default to "true." Value of "false" returns a "" string in log message.
dateTime	Date or time of log message	NA	enabled: Boolean whether to include date or time information in log message. Default is enabled. format: String of values to include in log message. Format of %y%m%d.%H%M%S.

Container	Container Description	Contained Containers	Contained Leaf and Description
dest	Holds destination logger's configuration settings.	console: Destination console. Only one allowed. file: destination file. Multiple allowed.	NA
console	Destination console	NA	enabled: Boolean that determines whether the console is enabled for logging. Default of "false."

Container	Container Description	Contained Containers	Contained Leaf and Description
file	Determines the settings of the destination file.	NA	enabled: Boolean that determines whether the destination is enabled. Default is "false." name: String of the destination log file. Default of "mtx-internal.log" location: String of destination file path. Default at "./mtxlogs." mbytes-rollover: uint32 that determines the length of the log file before the system overwrites the oldest data. Default is 10 Mbytes. hours-rollover: uint32 that determines the length of the log file in terms of hours. Default is 24 hours. startup-rollover: Boolean that determines if the log file is rolled over upon agent start or restart. Default value of "false." max-rollover-files: uint32 that determines the maximum number of rollover files; deletes the oldest file when the max-rollover-files value exceeded. Default value of 10.

Example

The following is the configuration file with the default installed configuration.

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="allActive" type="boolean" default="false">false</leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$">$DATETIME$ $COMPONENTID$ $TYPE$ $SRCFILE$ @ $SRCLINE$ $FCNINFO$:$MSG$</leaf>
```

```
        <container name="componentID">
               <leaf name="enabled" type="boolean" default="true"></leaf>
        </container>
        <container name="dateTime">
               <leaf name="enabled" type="boolean" default="true"></leaf>
               <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
        </container>
        <container name="fcn">
               <leaf name="enabled" type="boolean" default="true"></leaf>
               <leaf name="format" type="string"
   default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></leaf>
        </container>
         </container>
         <container name="dest">
           <container name="console">
             <leaf name="enabled" type="boolean" default="false">true</leaf>
           </container>
           <container name="file">
             <leaf name="enabled" type="boolean" default="false">true</leaf>
             <leaf name="name" type="string" default="mtx-internal.log"></leaf>
             <leaf name="location" type="string" default="./mtxlogs">/volatile</leaf>
       <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">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">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
         </list>
       </container>
     </container>
   </config>
```

CHAPTER 27

Model-Driven Telemetry

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 Nexus 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
  ```

  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 will appear 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 Telemetry

Telemetry has the following configuration guidelines and limitations:

- Telemetry is supported in Cisco NX-OS releases starting from 7.0(3)I7(1) for releases that support the data management engine (DME) Native Model.
- Release 7.0(3)I7(1) supports DME data collection, NX-API data sources, Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport, and JSON encoding over HTTP.
- The smallest sending interval (cadence) supported is five seconds for a depth of 0. The minimum cadence values for depth values greater than 0 depends on the size of the data being streamed out. Configuring cadences below the minimum value may result in undesirable system behavior.
- Up to five remote management receivers (destinations) are supported. Configuring more than five remote receivers may result in undesirable system behavior.
- In the event that a telemetry receiver goes down, other receivers will see data flow interrupted. The failed receiver must be restarted. Then start a new connection with the switch by unconfiguring then reconfiguring the failer receiver's IP address under the destination group.
- Telemetry can consume up to 20% of the CPU resource.
- To configure SSL certificate based authentication and the encryption of streamed data, you can provide a self signed SSL certificate with **certificate** *ssl cert path* **hostname "CN"** command. (NX-OS 7.0(3)I7(1) and later).

Configuration Commands After Downgrading to an Older Release

After a downgrade to an older release, some configuration commands or command options might fail because the older release may not support them. As a best practice when downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up to avoid 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
  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
  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 will disable the connection to the gRPC receiver if the gRPC receiver sends 20 errors. You will then need to unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

- The gRPC client sends the wrong certificate for secure connections,
- The gRPC receiver takes too long to handle client messages and incurs a timeout. Avoid timeouts by processing messages using a separate message processing thread.

Telemetry Compression for gRPC Transport

Starting with Cisco NX-OS 7.0(3)I7(1), 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 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
   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
```

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; i.e., show vlan id 100, show vlan id 101, etc.. Instead, use the CLI range options; i.e., show vlan id 100-110,204, whenever possible to improve performance.

     If only the summary/counter is needed, then avoid dumping a whole show command output to limit the bandwidth and data storage 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 5 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

Starting with Cisco NX-OS 7.0(3)I7(1), telemetry VRF support allows you to specify a transport VRF. This means that the telemetry data stream can egress via front-panel ports and avoid possible competition between SSH/NGINX control sessions.

You can use the **use-vrf** *vrf-name* command to specify the transport VRF.

The following example specifies the transport VRF:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-vrf test_vrf
```

The following is an example of use-vrf as a POST payload:

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptVrf": {
                      "attributes": {
                        "name": "default"
                      }
                    }
                  }
                ]
              }
            }
```

Configuring Telemetry Using the CLI

Configuring Telemetry Using the NX-OS CLI

The following steps enable streaming telemetry and configuring the source and destination of the data stream. These steps also include optional steps to enable and configure SSL/TLS certificates and GPB encoding.

Before you begin

Your switch must be running Cisco NX-OS Release 7.3(0)I5(1) or a later release.

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 will receive the data, where `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`
		• 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.
		Note After configuring the **use-vrf** command, you must configure a new destination IP address within the new VRF. However, you may re-use the same destination IP address by unconfiguring and reconfiguring the destination. This ensures that the telemetry data streams to the same destination IP address in the new VRF.

	Command or Action	Purpose

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 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*] **Example:** • The following command is applicable for DME, not for NX-API: `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"))`** • The following command is applicable for NX-API, not for DME: `switch(conf-tm-sensor)# ` **`path "show interface" depth 0`**	Add a sensor path to the sensor group. • 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 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**.

	Command or Action	Purpose
		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".

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	*ip_version* **address** *ip_address* **port** *portnum* **Example:** • For IPv4: `switch(conf-tm-dest)# **ip address 1.2.3.4 port 50003**`	Create a destination profile for the outgoing data. 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.

Step 15	**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.

	Command or Action	Purpose
Step 16	**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 17	**dst-grp** *dgrp_id* **Example:** `switch(conf-tm-sub)# dst-grp 100`	Link the destination group with ID *dgrp_id* to this subscription.

Configuration Examples for Telemetry Using the CLI

This example creates a subscription that streams data for the `sys/bgp` root MO every 5 seconds to the destination IP 1.2.3.4 port 50003.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data for `sys/intf` every 5 seconds to destination IP 1.2.3.4 port 50003, and encrypts the stream using GPB encoding 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 control database

This command displays the internal databases that reflect the configuration of telemetry.

```
switch# show telemetry control database ?
  <CR>
  >                   Redirect it to a file
  >>                  Redirect it to a file in append mode
  destination-groups  Show destination-groups
  destinations        Show destinations
  sensor-groups       Show sensor-groups
  sensor-paths        Show sensor-paths
  subscriptions       Show subscriptions
  |                   Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1

--------------------------------------------------------------------------------
Subscription ID      Data Collector Type
--------------------------------------------------------------------------------
100                  DME NX-API

Sensor Group Database size = 1
```

```
--------------------------------------------------------------------------------
Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
--------------------------------------------------------------------------------
100              Timer              10000(Running)         1

Sensor Path Database size = 1

--------------------------------------------------------------------------------
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
--------------------------------------------------------------------------------
No                       1              0           Full            sys/fm

Destination group Database size = 2

--------------------------------------------------------------------------------
Destination Group ID  Refcount
--------------------------------------------------------------------------------
100                   1

Destination Database size = 2

--------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
--------------------------------------------------------------------------------
192.168.20.111       12345      JSON       HTTP       1
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry control stats

This command displays the statistics about the internal databases about configuration of telemetry.

```
switch# show telemetry control stats
show telemetry control stats entered

--------------------------------------------------------------------------------
Error Description                                              Error Count
--------------------------------------------------------------------------------
Chunk allocation failures                                      0
Sensor path Database chunk creation failures                   0
Sensor Group Database chunk creation failures                  0
Destination Database chunk creation failures                   0
Destination Group Database chunk creation failures             0
Subscription Database chunk creation failures                  0
Sensor path Database creation failures                         0
Sensor Group Database creation failures                        0
Destination Database creation failures                         0
Destination Group Database creation failures                   0
Subscription Database creation failures                        0
Sensor path Database insert failures                           0
Sensor Group Database insert failures                          0
Destination Database insert failures                           0
Destination Group Database insert failures                     0
Subscription insert to Subscription Database failures          0
Sensor path Database delete failures                           0
Sensor Group Database delete failures                          0
Destination Database delete failures                           0
Destination Group Database delete failures                     0
Delete Subscription from Subscription Database failures        0
Sensor path delete in use                                      0
Sensor Group delete in use                                     0
Destination delete in use                                      0
Destination Group delete in use                                0
```

```
Delete destination(in use) failure count                    0
Failed to get encode callback                               0
Sensor path Sensor Group list creation failures             0
Sensor path prop list creation failures                     0
Sensor path sec Sensor path list creation failures          0
Sensor path sec Sensor Group list creation failures         0
Sensor Group Sensor path list creation failures             0
Sensor Group Sensor subs list creation failures             0
Destination Group subs list creation failures               0
Destination Group Destinations list creation failures       0
Destination Destination Groups list creation failures       0
Subscription Sensor Group list creation failures            0
Subscription Destination Groups list creation failures      0
Sensor Group Sensor path list delete failures               0
Sensor Group Subscriptions list delete failures             0
Destination Group Subscriptions list delete failures        0
Destination Group Destinations list delete failures         0
Subscription Sensor Groups list delete failures             0
Subscription Destination Groups list delete failures        0
Destination Destination Groups list delete failures         0
Failed to delete Destination from Destination Group         0
Failed to delete Destination Group from Subscription        0
Failed to delete Sensor Group from Subscription             0
Failed to delete Sensor path from Sensor Group              0
Failed to get encode callback                               0
Failed to get transport callback                            0
switch#  Destination Database size = 1

--------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
--------------------------------------------------------------------------------
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry data collector brief

This command displays the brief statistics about the data collection.

```
switch# show telemetry data collector brief

----------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
----------------------------------------------------------------------
DME                   143                         0
```

show telemetry data collector details

This command displays detailed statistics about the data collection which includes breakdown of all sensor paths.

```
switch# show telemetry data collector details

--------------------------------------------------------------------------------
Succ Collections      Failed Collections     Sensor Path
--------------------------------------------------------------------------------
150                   0                      sys/fm
```

show telemetry event collector errors

This command displays the errors statistic about the event collection.

```
switch# show telemetry event collector errors

--------------------------------------------------------------------------------
Error Description                                   Error Count
--------------------------------------------------------------------------------
APIC-Cookie Generation Failures                     - 0
Authentication Failures                             - 0
Authentication Refresh Failures                     - 0
Authentication Refresh Timer Start Failures         - 0
Connection Timer Start Failures                     - 0
Connection Attempts                                 - 3
Dme Event Subscription Init Failures                - 0
Event Data Enqueue Failures                         - 0
Event Subscription Failures                         - 0
Event Subscription Refresh Failures                 - 0
Pending Subscription List Create Failures           - 0
Subscription Hash Table Create Failures             - 0
Subscription Hash Table Destroy Failures            - 0
Subscription Hash Table Insert Failures             - 0
Subscription Hash Table Remove Failures             - 0
Subscription Refresh Timer Start Failures           - 0
Websocket Connect Failures                          - 0
```

show telemetry event collector stats

This command displays the statistics about the event collection which includes breakdown of all sensor paths.

```
switch# show telemetry event collector stats

--------------------------------------------------------------------------------
Collection Count  Latest Collection Time    Sensor Path
--------------------------------------------------------------------------------
```

show telemetry control pipeline stats

This command displays the statistics for the telemetry pipeline.

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
        Errors:
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0

    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
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
```

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
```

tmtrace.bin

This BASH shell command collects telemetry traces and prints them out.

```
switch# configure terminal
switch(config)# feature bash
switch(config)# run bash
```

```
bash-4.2$ tmtrace.bin -d tm-errors
bash-4.2$ tmtrace.bin -d tm-logs
bash-4.2$ tmtrace.bin -d tm-events
```

For example:

```
bash-4.2$ tmtrace.bin -d tm-logs
[01/25/17 22:52:24.563 UTC 1 29130] [3944724224][tm_ec_dme_auth.c:59] TM_EC: Authentication
 refresh url http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.565 UTC 2 29130] [3944724224][tm_ec_dme_rest_util.c:382] TM_EC: Performed
 POST request on http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.566 UTC 3 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER: Starting
 leaf timer for leaf:0x11e17ea4 time_in_ms:540000
[01/25/17 22:52:45.317 UTC 4 29130] [3944724224][tm_ec_dme_event_subsc.c:790] TM_EC: Event
 subscription database size 0
[01/25/17 22:52:45.317 UTC 5 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER: Starting
 leaf timer for leaf:0x11e17e3c time_in_ms:50000
bash-4.2#
```

Note The **tm-logs** option is not enabled by default because it is verbose.

Enable **tm-logs** with the `tmtrace.bin -L D tm-logs` command.

Disable **tm-logs** with the `tmtrace.bin -L W tm-logs` command.

show system internal telemetry trace

The **show system internal telemetry trace** [**tm-events** | **tm-errors** |**tm-logs** | **all**] command displays system internal telemetry trace information.

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy dest
 profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
1 does not have any sensor groups

3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#
switch# show system internal telemetry trace tm-events
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
```

```
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#
```

Configuring Telemetry Using the NX-API

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in the section "Telemetry Model in the DME." Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
    - **fmNxapi** — Contains the NX-API state.
    - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
    - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
        - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
        - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
    - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
        - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.
        - **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.
    - **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
        - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
        - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.
    - **telemetryCertificate** — Associates the telemetry subscription with a certificate and hostname.

To configure the telemetry feature using the NX-API, you must construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.

Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be running Cisco NX-OS Release 7.3(0)I5(1) or a later release.

Your switch must be configured to run the NX-API from the CLI:

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

NX-API sends telemetry data over management VRF:

```
switch(config)# nxapi use-vrf management
```

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`.

	Command or Action	Purpose

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**, or **NX-API**.
		Children of the sensor group object will include sensor paths and one or more relation objects (**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.

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": [`
	`         {`
		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

	Command or Action	Purpose

	```
	"telemetryDestOptSourceInterface": {
	                "attributes": {
	                   "name": "lo0"
	                }
	             }
	          }
	       ]
	    }
	}
	```
		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": {
	             "addr": "1.2.3.4",
	             "enc": "GPB",
	             "port": "50001",
	             "proto": "gRPC",
	             "rn":
	"addr-[1.2.3.4]-port-50001"
	        }
	    }
	}
	```
		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.
		• **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)

Step 8	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***.

	Command or Action	Purpose
		Children of the subscription object will include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).

Step 9	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 10	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 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**.

	Command or Action	Purpose
		• **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 11	Define one or more sensor paths or nodes to be monitored for telemetry.	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 regarding 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

Example:

Single sensor path

```
{
    "telemetrySensorPath": {
        "attributes": {
            "path": "sys/cdp",
            "rn": "path-[sys/cdp]",
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
    }
}
```

Example:

Single sensor path for NX-API

```
{
    "telemetrySensorPath": {
        "attributes": {
            "path": "show interface",
            "path": "show bgp",
            "rn": "path-[sys/cdp]",
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
    }
}
```

Example:

Multiple sensor paths

```
{
```

	Command or Action	Purpose

	```
	    "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"
	        }
	    }
	}
	```

Step 12	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
Step 13	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
Step 14	Add the destination group object as a child object to the root element (**telemetryEntity**).	

	Command or Action	Purpose

Step 15	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 16	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 17	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":`
	`"rsdestGroupRel-[sys/tm/dest-20]",`
	`        "tCl": "telemetryDestGroup",`
	`        "tDn": "sys/tm/dest-20",`
	`        "tType": "mo"`
	`    }`
	`}`
		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***]**.
		• **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.

	Command or Action	Purpose
		• **tType** — The target type, which is **mo** for managed object.
Step 18	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 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
```

DNs Available to Telemetry

For a list of DNs available to the telemetry feature, see Model-Driven Telemetry, on page 217.

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

CHAPTER 28

XML Management Interface

This section contains the following topics:

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 259 and RFC 4741.

For more information about using NETCONF over SSH, see RFC 4742.

This section includes the following topics:

NETCONF Layers

The following are the NETCONF layers:

Table 16: 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 259 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 17: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 269
commit	Sets the running configuration to the current contents of the candidate configuration.	NETCONF Commit Instance - Candidate Configuration Capability, on page 274
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 274
copy-config	Copies the content of source configuration datastore to the target datastore.	NETCONF copy-config Instance, on page 270
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 270 NETCONF rollback-on-error Instance , on page 274
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 259
get-config	Retrieves all or part of a configuration	NETCONF get-config Instance, on page 272
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 270

NETCONF Operation	Description	Example
lock	Allows the client to lock the configuration system of a device.	NETCONF Lock Instance, on page 272
unlock	Releases the configuration lock that the session issued.	NETCONF unlock Instance, on page 273
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	NETCONF validate Capability Instance , on page 275

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 258 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 259 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 268 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 18: 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

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 19: 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 269
- NETCONF Kill-session Instance, on page 270
- NETCONF copy-config Instance, on page 270
- NETCONF edit-config Instance, on page 270
- NETCONF get-config Instance, on page 272
- NETCONF Lock Instance, on page 272
- NETCONF unlock Instance, on page 273
- NETCONF Commit Instance - Candidate Configuration Capability, on page 274
- NETCONF Confirmed-commit Instance , on page 274
- NETCONF rollback-on-error Instance , on page 274
- NETCONF validate Capability Instance , on page 275

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)

APPENDIX A

Streaming Telemetry Sources

About Streaming Telemetry

The streaming telemetry feature of the Cisco Nexus 3000 and 9000 switches continuously streams data out of the network and notifies the client, providing near-real-time access to monitoring data.

Guidelines and Limitations

Following are the guideline and limitations for streaming telemetry:

- The telemetry feature is available in Cisco Nexus 3000 and 9000 switches.
- Switches with less than 8 GB of memory do not support telemetry.
- Software streaming telemetry does not support the TCP protocol. The tcp option is displayed in the Help text, but is not accepted during configuration.

Data Available for Telemetry

For each component group, the distinguished names (DNs) in the appendix of the NX-API DME Model Reference can provide the listed properties as data for telemetry.