Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 7.x

First Published: 2018-05-16

Last Modified: 2020-08-31

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

CONTENTS

PREFACE

Preface ix

Audience ix

Document Conventions ix

Related Documentation for Cisco Nexus 3000 Series Switches x

Documentation Feedback x

Communications, Services, and Additional Information x

CHAPTER 1

New and Changed Information 1

New and Changed Information 1

CHAPTER 2

Bash 3

About Bash 3

Accessing Bash 3

Escalate Privileges to Root 4

Examples of Bash Commands 5

Displaying System Statistics 5

Running Bash from CLI 5

Running Python from Bash 6

CHAPTER 3

Guest Shell 7

About the Guest Shell 7

Guidelines and Limitations 8

Accessing the Guest Shell 12

Resources Used for the Guest Shell 13

Capabilities in the Guest Shell 13

NX-OS CLI in the Guest Shell 14

Network Access in Guest Shell 14
Access to Bootflash in Guest Shell 16
Python in Guest Shell 17
Python 3 in Guest Shell 2.x (Centos 7) 17
Installing RPMs in the Guest Shell 20
Security Posture for Virtual Services 21
Digitally Signed Application Packages 21
Kernel Vulnerability Patches 22
ASLR and X-Space Support 22
Namespace Isolation 22
Root-User Restrictions 22
Resource Management 23
Guest File System Access Restrictions 23
Managing the Guest Shell 23
Disabling the Guest Shell 26
Destroying the Guest Shell 26
Enabling the Guest Shell 27
Replicating the Guest Shell 28
Exporting Guest Shell rootfs 28
Importing Guest Shell rootfs 28
Importing YAML File 29
show guestshell Command 33
Verifying Virtual Service and Guest Shell Information 33
Persistently Starting Your Application From the Guest Shell 35
Procedure for Persistently Starting Your Application from the Guest Shell 36
An Example Application in the Guest Shell 36

CHAPTER 4 **Python API** 39
About the Python API 39
Using Python 39
Cisco Python Package 39
Using the CLI Command APIs 40
Invoking the Python Interpreter from the CLI 42
Display Formats 42

Non-Interactive Python 44
Running Scripts with Embedded Event Manager 45
Python Integration with Cisco NX-OS Network Interfaces 46
Cisco NX-OS Security with Python 46
Examples of Security and User Authority 47
Example of Running Script with Scheduler 48

CHAPTER 5 **Scripting with Tcl** 49
About Tcl 49
Tclsh Command Help 49
Tclsh Command History 50
Tclsh Tab Completion 50
Tclsh CLI Command 50
Tclsh Command Separation 50
Tcl Variables 51
Tclquit 51
Tclsh Security 51
Running the Tclsh Command 51
Navigating Cisco NX-OS Modes from the Tclsh Command 52
Tcl References 54

CHAPTER 6 **Ansible** 55
Prerequisites 55
About Ansible 55
Cisco Ansible Module 55

CHAPTER 7 **Puppet Agent** 57
About Puppet 57
Prerequisites 57
Puppet Agent NX-OS Environment 58
ciscopuppet Module 58

CHAPTER 8 **Using Chef Client with Cisco NX-OS** 61
About Chef 61

Prerequisites 61
Chef Client NX-OS Environment 62
cisco-cookbook 62

CHAPTER 9 NX-API 65
About NX-API 65
Feature NX-API 65
Transport 66
Message Format 66
Security 66
Guidelines and Limitations 66
Using NX-API 67
NX-API Management Commands 69
Working With Interactive Commands Using NX-API 70
NX-API Request Elements 70
NX-API Response Elements 73
About JSON (JavaScript Object Notation) 74
CLI Execution 74
JSON-RPC, JSON, and XML Supported Commands 74
Examples of XML and JSON Output 75

CHAPTER 10 NX-API Response Codes 79
Table of NX-API Response Codes 79

CHAPTER 11 XML Support for ABM and LM in N3500 81
XML Support for ABM and LM in N3500 81

CHAPTER 12 Model-Driven Telemetry 89
About Telemetry 89
Telemetry Components and Process 89
High Availability of the Telemetry Process 91
Licensing Requirements for Telemetry 91
Installing and Upgrading Telemetry 91
Guidelines and Limitations for Model-Driven Telemetry 92

Configuring Telemetry Using the CLI 96
Configuring Streaming Telemetry Using the Cisco NX-OS CLI 96
Configuration Examples for Telemetry Using the CLI 100
Displaying Telemetry Configuration and Statistics 104
Displaying Telemetry Log and Trace Information 109
Configuring Telemetry Using the NX-API 110
Configuring Telemetry Using the NX-API 110
Configuration Example for Telemetry Using the NX-API 119
Telemetry Model in the DME 122

CHAPTER 13 XML Management Interface 125
About the XML Management Interface 125
About the XML Management Interface 125
NETCONF Layers 125
SSH xmlagent 126
Licensing Requirements for the XML Management Interface 126
Prerequisites to Using the XML Management Interface 127
Using the XML Management Interface 127
Configuring SSH and the XML Server Options 127
Starting an SSH Session 127
Sending the Hello Message 128
Obtaining the XSD Files 128
Sending an XML Document to the XML Server 129
Creating NETCONF XML Instances 129
RPC Request Tag rpc 130
NETCONF Operations Tags 131
Device Tags 132
Extended NETCONF Operations 134
NETCONF Replies 137
RPC Response Tag 138
Interpreting Tags Encapsulated in the Data Tag 138
Information About Example XML Instances 139
Example XML Instances 139
NETCONF Close Session Instance 139

NETCONF Kill-session Instance 140
NETCONF copy-config Instance 140
NETCONF edit-config Instance 140
NETCONF get-config Instance 142
NETCONF Lock Instance 142
NETCONF unlock Instance 143
NETCONF Commit Instance - Candidate Configuration Capability 144
NETCONF Confirmed-commit Instance 144
NETCONF rollback-on-error Instance 144
NETCONF validate Capability Instance 145
Additional References 145

CHAPTER 14 **Converting CLI Commands to Network Configuration Format** 147
Information About XMLIN 147
Licensing Requirements for XMLIN 147
Installing and Using the XMLIN Tool 148
Converting Show Command Output to XML 148
Configuration Examples for XMLIN 149

Preface

This preface includes the following sections:

- Audience, on page ix
- Document Conventions, on page ix
- Related Documentation for Cisco Nexus 3000 Series Switches, on page x
- Documentation Feedback, on page x
- Communications, Services, and Additional Information, on page x

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.

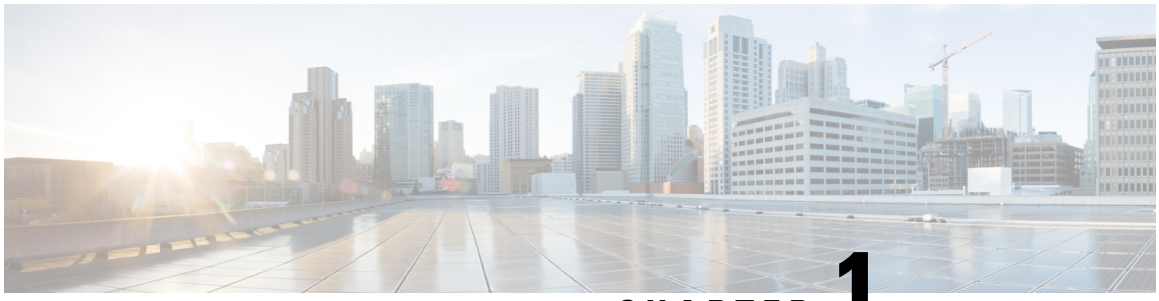
Communications, Services, and Additional Information

- 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.

Preface



CHAPTER 1

New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 7.x*.

- New and Changed Information, on page 1

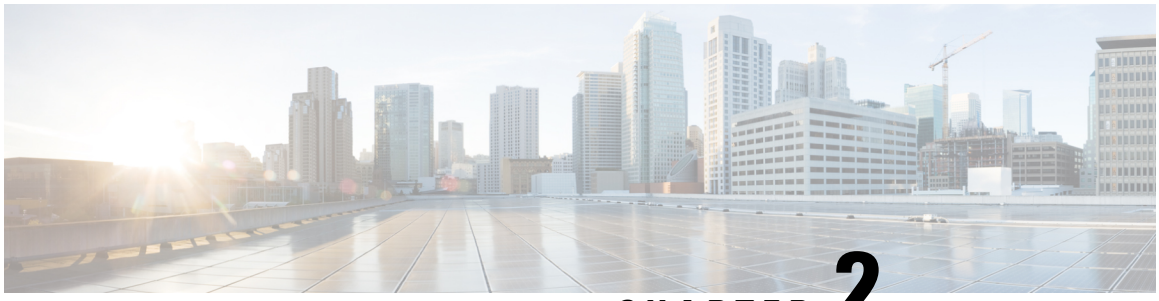
New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 7.x* and where they are documented.

Table 1: New and Changed Features

Feature	Description	Changed in Release	Where Documented
Model-Driven Telemetry	Support added for Cisco Nexus 3500 platform switches.	7.0(3)I7(9)	Model-Driven Telemetry, on page 89
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 147
XML Management Interface	Support for managing the Cisco Nexus 3500 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 125

New and Changed Information



CHAPTER 2

Bash

- About Bash, on page 3
- Accessing Bash, on page 3
- Escalate Privileges to Root, on page 4
- Examples of Bash Commands, on page 5

About Bash

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

Accessing Bash

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

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

```
switch# show role name dev-ops

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

switch# show role name network-admin

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

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

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

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

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

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

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

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

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

```
run bash whoami
```

Escalate Privileges to Root

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

The following are guidelines for escalating privileges:

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

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

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

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

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

Password:

Linux# whoami
```

```
root
Linux# exit
exit
```

Examples of Bash Commands

This section contains examples of Bash commands and output.

Displaying System Statistics

The following example shows how to display system statistics:

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

Running Bash from CLI

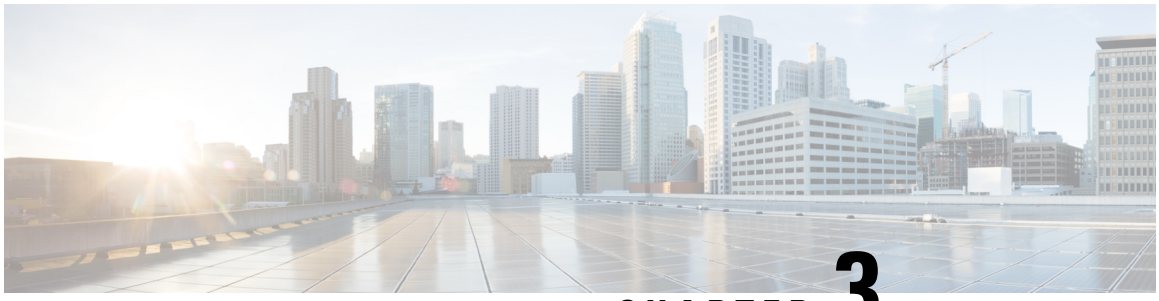
The following example shows how to run a bash command from the CLI with the **run bash** <*command*> command:

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

...

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

Running Python from Bash

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

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

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

version 6.0(2)U4(1)

interface Ethernet1/3
  no switchport
  vrf member myvrf


>>>
```



CHAPTER 3

Guest Shell

- About the Guest Shell, on page 7
- Guidelines and Limitations, on page 8
- Accessing the Guest Shell, on page 12
- Resources Used for the Guest Shell, on page 13
- Capabilities in the Guest Shell, on page 13
- Security Posture for Virtual Services, on page 21
- Guest File System Access Restrictions , on page 23
- Managing the Guest Shell, on page 23
- Verifying Virtual Service and Guest Shell Information, on page 33
- Persistently Starting Your Application From the Guest Shell, on page 35
- Procedure for Persistently Starting Your Application from the Guest Shell, on page 36
- An Example Application in the Guest Shell, on page 36

About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, switches support access to a decoupled execution space running within a Linux Container (LXC) called the “Guest Shell”.

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

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

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

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

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

Note Beginning with Cisco NX-OS 7.0(3)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 switch is in a network that uses an HTTP proxy server, the **http_proxy** and **https_proxy** environment variables must be set up within the Guest Shell also.

Example:

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

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

Access to Bootflash in Guest Shell

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

Example:

```
find . -name "foo.txt"
rm "/bootflash/junk/foo.txt"
```

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/df50ac98ea0eb006c55a399c3bf1db9da7b5a24de7890bc9cfd5dd9e99/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 switches are only two of the many ways that the network-admin can manage or extend the functionality of the system. These options are geared toward providing an execution environment that is decoupled from the native host context. This separation allows the introduction of software into the system that may not be compatible with the native execution environment. It also allows the software to run in an environment that does not interfere with the behavior, performance, or scale of the system.

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 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 2: Guest Shell CLI Commands

Commands	Description
guestshell enable {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• When *guest shell OVA file* is specified: Installs and activates the Guest Shell using the OVA that is embedded in the system image. Installs and activates the Guest Shell using the specified software package (OVA file) or the embedded package from the system image (when no package is specified). Initially, Guest Shell packages are only available by being embedded in the system image. When the Guest Shell is already installed, this command enables the installed Guest Shell. Typically this is used after a **guestshell disable** command. • When *rootfs-file-URI* is specified: Imports a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.
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        guestshe11.ova
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n) y

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

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

Destroying the Guest Shell

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

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

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

switch# guestshell destroy
```

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

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

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

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

Enabling the Guest Shell

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

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

Example:

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

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

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

Replicating the Guest Shell

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

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

Exporting Guest Shell rootfs

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

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

The **guestshell export rootfs package** command:

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

Importing Guest Shell rootfs

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

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

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

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

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

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

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

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

Note To restore the embedded version of the rootfs:

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

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

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

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

Example workflow for **guestshell enable** :

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

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

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

Importing YAML File

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

complete descriptor for the Guest Shell container. It only contains some of the parameters that are user modifiable.

Example of a Guest Shell import YAML file:

```
---
import-schema-version: "1.0"
info:
  name: "GuestShell"
  version: "2.2(0.3)"
  description: "Exported GuestShell: 20170216T175137Z"
app:
  apptype: "lxc"
  cpuarch: "x86_64"
  resources:
    cpu: 3
    memory: 307200
    disk:
    - target-dir: "/"
      capacity: 250
...
```

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

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

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

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

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

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

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

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

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Guest Shell import schema",
  "description": "Schema for Guest Shell import descriptor file - ver 1.0",
  "copyright": "2017 by Cisco systems, Inc. All rights reserved.",
  "id": "",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "import-schema-version": {
      "id": "/import-schema-version",
```

```
  "type": "string",
  "minLength": 1,
  "maxLength": 20,
  "enum": [
      "1.0"
  ]
},
"info": {
  "id": "/info",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "name": {
      "id": "/info/name",
      "type": "string",
      "minLength": 1,
      "maxLength": 29
    },
    "description": {
      "id": "/info/description",
      "type": "string",
      "minLength": 1,
      "maxLength": 199
    },
    "version": {
      "id": "/info/version",
      "type": "string",
      "minLength": 1,
      "maxLength": 63
    },
    "author-name": {
      "id": "/info/author-name",
      "type": "string",
      "minLength": 1,
      "maxLength": 199
    },
    "author-link": {
      "id": "/info/author-link",
      "type": "string",
      "minLength": 1,
      "maxLength": 199
    }
  }
},
"app": {
  "id": "/app",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "apptype": {
      "id": "/app/apptype",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
        "lxc"
      ]
    },
    "cpuarch": {
      "id": "/app/cpuarch",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
```

```
        "x86_64"
      ]
    },
    "resources": {
      "id": "/app/resources",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "cpu": {
          "id": "/app/resources/cpu",
          "type": "integer",
          "multipleOf": 1,
          "maximum": 100,
          "minimum": 1
        },
        "memory": {
          "id": "/app/resources/memory",
          "type": "integer",
          "multipleOf": 1024,
          "minimum": 1024
        },
        "disk": {
          "id": "/app/resources/disk",
          "type": "array",
          "minItems": 1,
          "maxItems": 1,
          "uniqueItems": true,
          "items": {
            "id": "/app/resources/disk/0",
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "target-dir": {
                "id": "/app/resources/disk/0/target-dir",
                "type": "string",
                "minLength": 1,
                "maxLength": 1,
                "enum": [
                  "/"
                ]
              },
              "file": {
                "id": "/app/resources/disk/0/file",
                "type": "string",
                "minLength": 1,
                "maxLength": 63
              },
              "capacity": {
                "id": "/app/resources/disk/0/capacity",
                "type": "integer",
                  "multipleOf": 1,
                  "minimum": 1
              }
            }
          }
        }
      },
      "required": [
        "memory",
        "disk"
      ]
    }
  },
  "required": [
```

```
        "apptype",
        "cpuarch",
        "resources"
      ]
    }
  },
  "required": [
    "app"
  ]
}
```

show guestshell Command

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

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

```
switch# show guestshell detail
Virtual service guestshell+ detail
  State                 : Activated
  Package information
    Name                : rootfs_puppet
    Path                : usb2:/rootfs_puppet
    Application
      Name              : GuestShell
      Installed version : 2.3(0.0)
      Description       : Exported GuestShell: 20170613T173648Z
    Signing
      Key type          : Unsigned
      Method            : Unknown
    Licensing
      Name              : None
      Version           : None
```

Verifying Virtual Service and Guest Shell Information

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

Command	Description

show virtual-service global
`switch# show virtual-service global`
 	
`Virtual Service Global State and Virtualization Limits:`
 	
`Infrastructure version : 1.9`
`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.

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

Command	Description

show guestshell detail
 	
`switch# show guestshell detail`
`Virtual service guestshell+ detail`
`  State                 : Activated`
`  Package information`
`    Name                : guestshell.ova`
`    Path                : /isan/bin/guestshell.ova`
`    Application`
`      Name              : GuestShell`
`      Installed version : 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
```

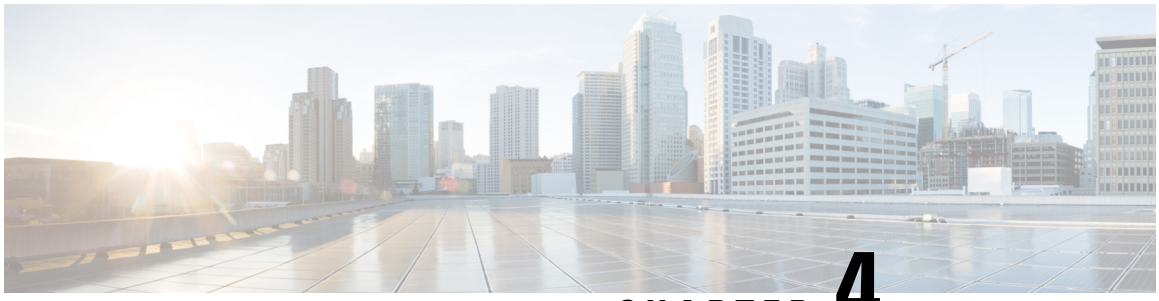
```
            ##355 /bin/bash /etc/init.d/hello.sh &
            ##367 sleep 10

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 4

Python API

- **About the Python API** , on page 39
- Using Python, on page 39

About the Python API

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

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

http://www.python.org/

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

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

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

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

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

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

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

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

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

NAME
    cisco

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

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

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

Using the CLI Command APIs

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

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

Table 3: CLI Command APIs

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

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

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

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

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

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

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

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

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

Invoking the Python Interpreter from the CLI

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

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

```
switch# python
switch# python

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

Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print intf['interface']
...
mgmt0
loopback1
>>>
```

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

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

```
switch# show file bootflash: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 the pipe operator ("|").

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

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

Running Scripts with Embedded Event Manager

On Cisco Nexus switches, Embedded Event Manager (EEM) policies support Python scripts.

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

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

```
switch# show running-config eem

!Command: show running-config eem
!Running configuration last done at: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

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


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

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

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

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

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
```

```
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
Python

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

PC_VSH_CMD_TLV(7679) with q
```

Python Integration with Cisco NX-OS Network Interfaces

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

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

```
switch# python
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()
```

Python 3 example.

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')
'Warning: No NTP peer/server configured. Time may be out of sync.\n15:39:39.513 UTC Thu Jun
 25 2020\nTime source is NTP\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

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

version 9.3(5) Bios:version 07.67

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

```
vrf context myvrf
```

The following is an example for a nonprivileged user:

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

The following example shows an RBAC configuration:

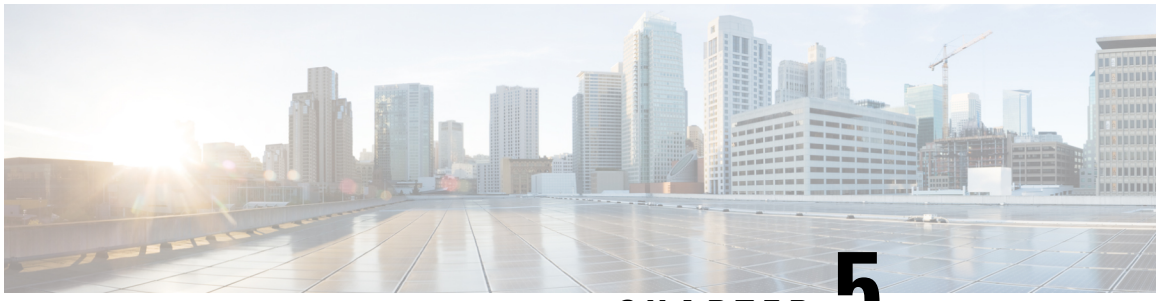
```
switch# show user-account
user:admin
        this user account has no expiry date
        roles:network-admin
user:pyuser
        this user account has no expiry date
        roles:network-operator python-role
switch# show role name python-role
```

Example of Running Script with Scheduler

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

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

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

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

Python 3 example.



CHAPTER 5

Scripting with Tcl

- About Tcl, on page 49
- Running the Tclsh Command, on page 51
- Navigating Cisco NX-OS Modes from the Tclsh Command, on page 52
- Tcl References, on page 54

About Tcl

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

This section describes how to run Tcl scripts or run Tcl interactively on switches.

Tclsh Command Help

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

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

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

switch-tcl#
```

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

Tclsh Command History

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

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

Tclsh Tab Completion

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

Tclsh CLI Command

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

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

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

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

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

If you use the following commands in your script, the script fails and the Tcl shell displays an error:

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

Tclsh Command Separation

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

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

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

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

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

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

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

Tcl Variables

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

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

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

Tclquit

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

Tclsh Security

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

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

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

Running the Tclsh Command

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

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

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

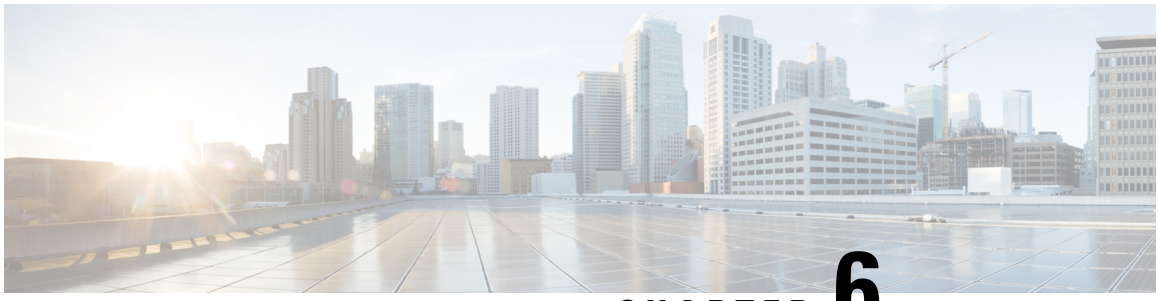
The following example shows how to change Cisco NX-OS modes from an interactive Tcl shell:

```
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 6

Ansible

- Prerequisites, on page 55
- About Ansible, on page 55
- Cisco Ansible Module, on page 55

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:

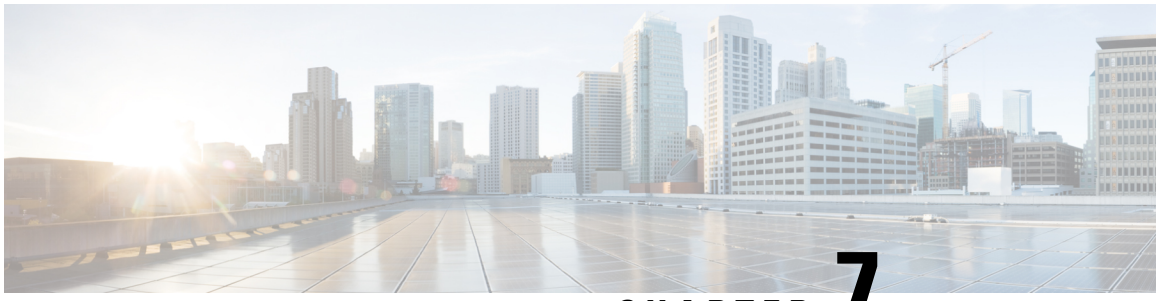
Ansible	https://www.ansible.com/
Ansible Automation Solutions. Includes installation instructions, playbook instructions and examples, module lists, and so on.	https://docs.ansible.com/

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 7

Puppet Agent

This chapter includes the following sections:

- About Puppet, on page 57
- Prerequisites, on page 57
- Puppet Agent NX-OS Environment, on page 58
- ciscopuppet Module, on page 58

About Puppet

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

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

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

More information can be found from Puppet Labs:

Puppet Labs	https://puppetlabs.com
Puppet Labs FAQ	https://puppet.com/products/faq
Puppet Labs Documentation	https://puppet.com/docs

Prerequisites

The following are prerequisites for the Puppet Agent:

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

- Cisco Nexus 3600 platform switches.
- Cisco Nexus 3500 platform switches
- Cisco Nexus 3100 platform switches.
- Cisco Nexus 3000 Series switches.
- Cisco NX-OS Release 7.0(3)I2(1) or later.

- You must have the required disk storage available on the device for virtual services installation and deployment of Puppet Agent.
  - A minimum of 450MB free disk space on bootflash.
- You must have Puppet Primary server with Puppet 4.0 or later.
- You must have Puppet Agent 4.0 or later.

Puppet Agent NX-OS Environment

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

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

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

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

ciscopuppet Module

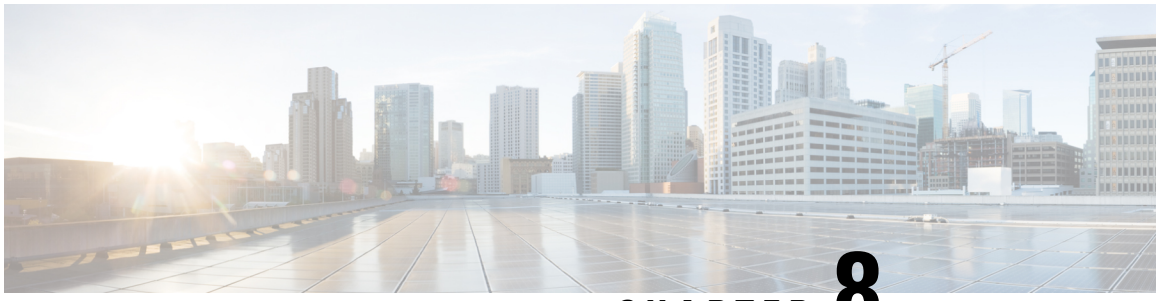
The ciscopuppet module is a Cisco developed open-source software module. It interfaces between the abstract resources configuration in a puppet manifest and the specific implementation details of the Cisco NX-OS operating system and platform. This module is installed on the Puppet Primary and is required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

The ciscopuppet module is available on Puppet Forge.

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

ciscopuppet Module location (Puppet Forge)	Puppet Forge
Resource Type Catalog	Cisco Puppet Resource Reference
ciscopuppet Module: Source Code Repository	Cisco Network Puppet Module

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



CHAPTER 8

Using Chef Client with Cisco NX-OS

This chapter includes the following sections:

- About Chef, on page 61
- Prerequisites, on page 61
- Chef Client NX-OS Environment, on page 62
- cisco-cookbook, on page 62

About Chef

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

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

The following references provide more information from Chef:

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

Chef Client NX-OS Environment

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

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

The workflow for both use cases is similar.

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

Topic	Link
Chef Client (Native)	Latest information on Client RPM is available here.
Chef Client (Guest Shell, CentOs7)	Latest information on Client RPM is available here.
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

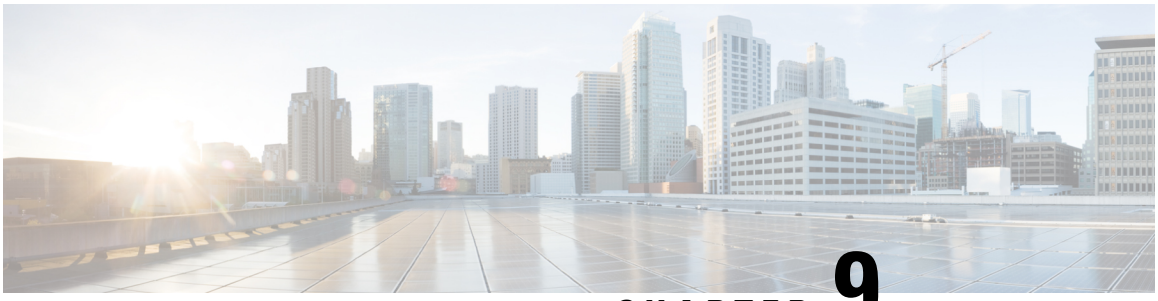
cisco-cookbook

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 operating system and Cisco Nexus switches. This cookbook is installed on the Chef Server and is required for proper Chef Client operation on Cisco Nexus switches.

cisco-cookbook can be found on Chef Supermarket.

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

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



CHAPTER 9

NX-API

- About NX-API, on page 65
- Guidelines and Limitations, on page 66
- Using NX-API, on page 67
- JSON-RPC, JSON, and XML Supported Commands, on page 74

About NX-API

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

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

Feature NX-API

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

```
BLR-VXLAN-NPT-CR-179# show nxapi
 nxapi enabled
 HTTP Listen on port 80
 HTTPS Listen on port 443
 BLR-VXLAN-NPT-CR-179#
```

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

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

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

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

Transport

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

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

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

Message Format

Note

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

Security

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

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

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

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

Guidelines and Limitations

On the Cisco Nexus 3548, NX-API has the following limitation:

- Scripted NX-API polling of **show hardware profile buffer monitor detail last 300** for multiple iterations and parallel sessions can possibly cause slow responses. Polling does complete successfully, but numerous polling iterations of this CLI from multiple parallel sessions occasionally delay in the return of the CLI output.

Using NX-API

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

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

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

- Enable the management interface.

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

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

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

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

Request:

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

Response:

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

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

Request:

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

Response:

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

Using the Management Interface for NX-API calls

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

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

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

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

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

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

```
  class copp-http
    police pps 100
!
```

NX-API Management Commands

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

Table 4: NX-API Management Commands

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

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)#
```

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

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

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

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

```
switch(config)#
```

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

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

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

See Guest Shell, on page 7 for more information about Guest Shell.

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

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

Working With Interactive Commands Using NX-API

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

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

```
terminal dont-ask ; reload module 21
```

```
terminal dont-ask ; system mode maintenance
```

NX-API Request Elements

NX-API request elements are sent to the switch in XML format or JSON format. The HTTP header of the request must identify the content type of the request.

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

Table 5: NX-API Request Elements

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

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

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

Enable or disable chunk with the following settings:

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

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

You can use XML SAX to parse the chunked output.

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

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

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

The following are examples of multiple commands:

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

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>output_format</td><td>The available output message formats are the following:
<table>
<tr><td>xml</td><td>Specifies output in XML format.</td></tr>
<tr><td>json</td><td>Specifies output in JSON format.</td></tr>
</table>
Note The Cisco Nexus 3500 platform switches CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.

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

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

NX-API Response Elements

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

Table 6: NX-API Response Elements

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

NX-API Response Element	Description
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.

About JSON (JavaScript Object Notation)

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

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

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

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

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

CLI Execution

Show_Command | json

Example Code

```
BLR-VXLAN-NPT-CR-179# show cdp neighbors  | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "
83886080", "device_id": "SW-SPARSHA-SAVBU-F10", "intf_id": "mgmt0", "ttl": "148"
, "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco WS-C2960
S-48TS-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device
_id": "BLR-VXLAN-NPT-CR-178(FOC1745R01W)", "intf_id": "Ethernet1/1", "ttl": "166
", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Disput
e"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

JSON-RPC, JSON, and XML Supported Commands

Starting with Cisco NX-OS Release 6.0(2)U4(1), JSON-RPC, JSON, and XML output of the following commands is supported:

- show bgp all

- show bgp process
- show bgp convergence
- show bgp ip unicast/multicast
- show bgp ipv4 unicast/multicast
- show bgp ipv6 unicast/multicast
- show bgp paths
- show bgp peer-policy
- show bgp vrf
- show bgp sessions
- show bgp statistics
- show consistency-checker forwarding ipv4
- show consistency-checker forwarding ipv6
- show lldp neighbors
- show lldp neighbors detail
- show lldp neighbors interface ethernet x/x
- show lldp neighbors interface ethernet x/x detail
- show lldp portid-subtype
- show lldp timers
- show lldp tlv-select
- show lldp traffic
- show lldp traffic interface ethernet x/x
- show process memory
- show process cpu & show process
- show routing vrf all
- show system internal forwarding route summary
- show system resources

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:

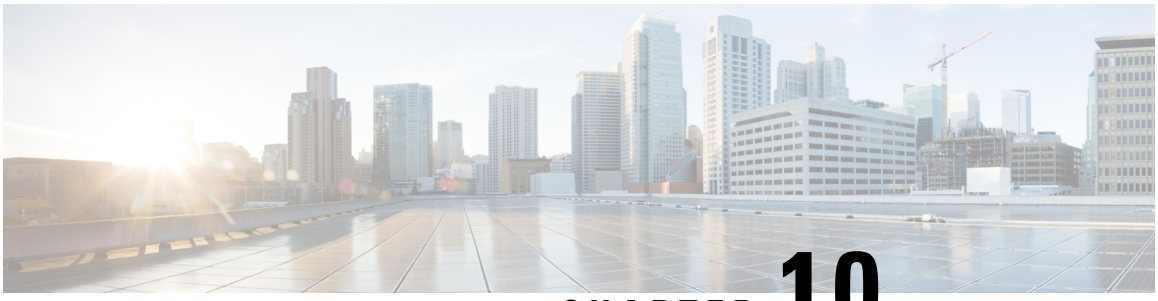
```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:fib">
 <nf:data>
  <show>
   <hardware>
    <profile>
     <status>
      <__XML__OPT_Cmd_dynamic_tcam_status>
       <__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 10

NX-API Response Codes

- Table of NX-API Response Codes, on page 79

Table of NX-API Response Codes

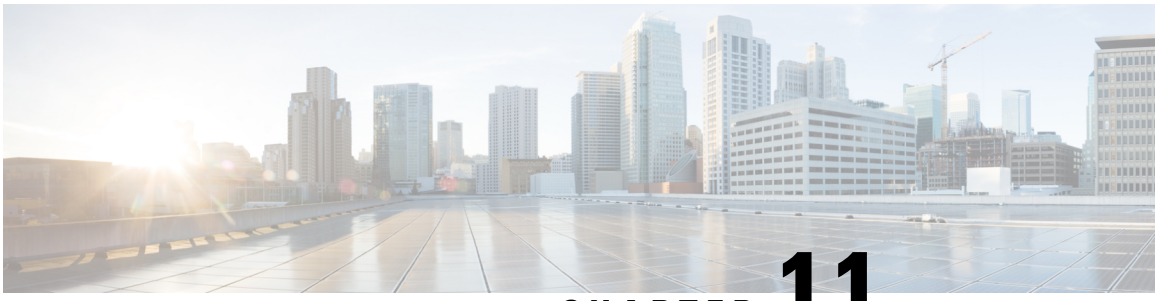
The following are the possible NX-API errors, error codes, and messages of an NX-API response.

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 7: NX-API Response Codes

NX-API Response	Code	Message
SUCCESS	200	Success.
CUST_OUTPUT_PIPED	204	Output is piped elsewhere due to request.
BASH_CMD_ERR	400	Input Bash command error.
CHUNK_ALLOW_ONE_CMD_ERR	400	Chunking only allowed to one command.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.
CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow **show** .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive show commands exceeded. The maximum is 10.

MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Backend processing error.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.



CHAPTER 11

XML Support for ABM and LM in N3500

• XML Support for ABM and LM in N3500 , on page 81

XML Support for ABM and LM in N3500

The following commands show XML Output for ABM and LM:

show hardware profile buffer monitor sampling

CLI :

```
MTC-8(config)# show hardware profile buffer monitor sampling

Sampling CLI issued at: 05/25/2016 04:18:56

Sampling interval: 200
```

XML :

```
MTC-8(config)# show hardware profile buffer monitor sampling | xml

<?xml version="1.0" encoding="ISO-8859-1"?>

<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w

ww.cisco.com/nxos:1.0:mtc_usd_cli">

 <nf:data>

  <show>

   <hardware>

    <profile>

     <buffer>

      <monitor>

       <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>

        <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>

         <__readonly__>

          <cmd_name>Sampling CLI</cmd_name>
```

XML Support for ABM and LM in N3500

```
            <cmd_issue_time>05/25/2016 04:19:12</cmd_issue_time>
            <TABLE_sampling>
             <ROW_sampling>
              <sampling_interval>200</sampling_interval>
             </ROW_sampling>
            </TABLE_sampling>
           </__readonly__>
          </__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
         </__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
        </monitor>
       </buffer>
      </profile>
     </hardware>
    </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```

show hardware profile buffer monitor detail | xml

XML :

```
<show>
   <hardware>
    <profile>
     <buffer>
      <monitor>
       <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
        <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
         <__readonly__>
          <cmd_name>Detail CLI</cmd_name>
          <cmd_issue_time>10/02/2001 10:58:58</cmd_issue_time>
          <TABLE_detail_entry>
           <ROW_detail_entry>
            <detail_util_name>Ethernet1/1</detail_util_name>
            <detail_util_state>Active</detail_util_state>
           </ROW_detail_entry>
           <ROW_detail_entry>
            <time_stamp>10/02/2001 10:58:58</time_stamp>
            <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
            <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
            <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
            <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
            <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
            <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
            <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
            <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
            <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
```

XML Support for ABM and LM in N3500

```
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:57</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:56</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:55</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
```

```
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:54</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
```

show hardware profile buffer monitor brief

XML :

```
show hardware profile buffer monitor brief | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:mtc_usd_cli">
<nf:data>
<show>
<hardware>
<profile>
<buffer>
<monitor>
<__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
<__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
<__readonly__>
<cmd_name>Brief CLI</cmd_name>
<cmd_issue_time>03/21/2016 09:06:38</cmd_issue_time>
<TABLE_ucst_hdr>
<ROW_ucst_hdr>
<ucst_hdr_util_name>Buffer Block 1</ucst_hdr_util_name>
<ucst_hdr_1sec_util>0KB</ucst_hdr_1sec_util>
<ucst_hdr_5sec_util>0KB</ucst_hdr_5sec_util>
<ucst_hdr_60sec_util>N/A</ucst_hdr_60sec_util>
<ucst_hdr_5min_util>N/A</ucst_hdr_5min_util>
<ucst_hdr_1hr_util>N/A</ucst_hdr_1hr_util>
<ucst_hdr_total_buffer>Total Shared Buffer Available = 5397 Kbytes
</ucst_hdr_total_buffer>
<ucst_hdr_class_threshold>Class Threshold Limit = 5130 Kbytes
</ucst_hdr_class_threshold>
</ROW_ucst_hdr>
</TABLE_ucst_hdr>
<TABLE_brief_entry>
<ROW_brief_entry>
<brief_util_name>Ethernet1/45</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>

<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/46</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
```

```
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/47</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/48</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/21</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/22</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/23</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>

<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/24</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/9</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/10</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/11</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/12</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
```

XML Support for ABM and LM in N3500

```
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
```

show hardware profile latency monitor sampling

CLI

```
MTC-8(config)# show hardware profile latency monitor sampling

Sampling CLI issued at: 05/25/2016 04:19:54

Sampling interval: 20
```

XML

```
MTC-8(config)# show hardware profile latency monitor sampling | xml

<?xml version="1.0" encoding="ISO-8859-1"?>

<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w

ww.cisco.com/nxos:1.0:mtc_usd_cli">

 <nf:data>

  <show>

   <hardware>

    <profile>

     <latency>

      <monitor>

       <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

        <__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>

         <__readonly__>

          <cmd_issue_time>05/25/2016 04:20:06</cmd_issue_time>

          <device_instance>0</device_instance>

          <TABLE_sampling>

           <ROW_sampling>

            <sampling_interval>20</sampling_interval>

           </ROW_sampling>

          </TABLE_sampling>

         </__readonly__>

        </__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>

       </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

      </monitor>

     </latency>
```

```
    </profile>
   </hardware>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```

show hardware profile latency monitor threshold

CLI

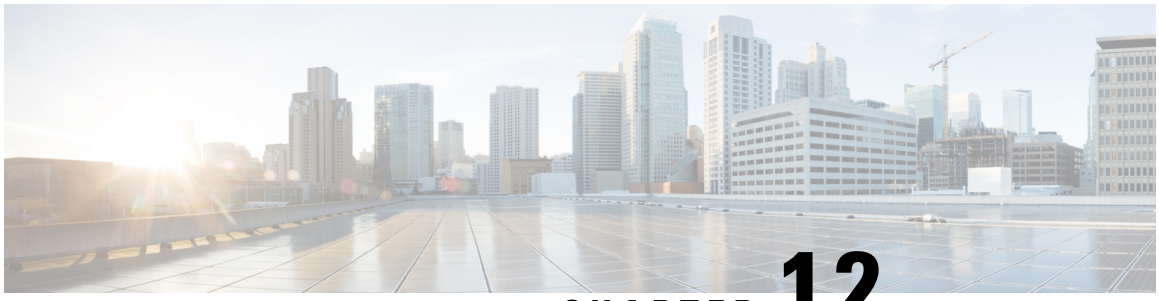
```
MTC-8(config)# show hardware profile latency monitor threshold
Sampling CLI issued at: 05/25/2016 04:20:53
Threshold Avg: 3000
Threshold Max: 300000
```

XML

```
MTC-8(config)# show hardware profile latency monitor threshold | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:mtc_usd_cli">
 <nf:data>
  <show>
   <hardware>
    <profile>
     <latency>
      <monitor>
       <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
        <__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>
         <__readonly__>
          <cmd_issue_time>05/25/2016 04:21:04</cmd_issue_time>
          <device_instance>0</device_instance>
          <TABLE_threshold>
           <ROW_threshold>
            <threshold_avg>3000</threshold_avg>
            <threshold_max>300000</threshold_max>
           </ROW_threshold>
```

XML Support for ABM and LM in N3500

```
          </TABLE_threshold>
         </__readonly__>
        </__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>
       </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
      </monitor>
     </latency>
    </profile>
   </hardware>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```



CHAPTER 12

Model-Driven Telemetry

- About Telemetry, on page 89
- Licensing Requirements for Telemetry, on page 91
- Installing and Upgrading Telemetry, on page 91
- Guidelines and Limitations for Model-Driven Telemetry, on page 92
- Configuring Telemetry Using the CLI, on page 96
- Configuring Telemetry Using the NX-API, on page 110

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.
- **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 that is not included in a license package is bundled with the Cisco NX-OS image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the *Cisco NX-OS Licensing Guide*.

Installing and Upgrading Telemetry

Installing the Application

The telemetry application is packaged as a feature RPM and included with the NX-OS release. The RPM is installed by default as part of the image bootup. After installation, you can start the application using the **feature telemetry** command. The RPM file is located in the `/rpms` directory and is named as follows:

telemetry-*version-build_ID***.libn32_n3000.rpm**

As in the following example:

```
telemetry-2.0.0-7.0.3.I5.1.lib32_n3000.rpm
```

Installing Incremental Updates and Fixes

Copy the RPM to the device bootflash and use the following commands from the `bash` prompt:

```
feature bash
run bash sudo su
```

Then copy the RPM to the device bootflash. Use the following commands from the `bash` prompt:

yum upgrade *telemetry_new_version*.**rpm**

The application is upgraded and the change appears when the application is started again.

Downgrading to a Previous Version

To downgrade the telemetry application to a previous version, use the following command from the `bash` prompt:

```
yum downgrade telemetry
```

Verifying the Active Version

To verify the active version, run the following command from the switch `exec` prompt:

```
show install active
```

Note The `show install active` command will only show the active installed RPM after an upgrade has occurred. The default RPM that comes bundled with the NX-OS will not be displayed.

Guidelines and Limitations for Model-Driven Telemetry

Telemetry has the following configuration guidelines and limitations:

- Telemetry is supported in Cisco NX-OS releases starting from 7.0(3)I5(1) for releases that support the data management engine (DME) Native Model.
- Release 7.0(3)I6(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.
- If a telemetry receiver goes down, other receivers see data flow interrupted. The failed receiver must be restarted. Then start a new connection with the switch by unconfiguring then reconfiguring the failed receiver's IP address under the destination group.
- Telemetry can consume up to 20% of the CPU resource.
- To configure SSL certificate-based authentication and the encryption of streamed data, you can provide a self-signed SSL certificate with **certificate** *ssl cert path* **hostname "CN"** command. (NX-OS 7.0(3)I7(1) and later).
- QoS Explicit Congestion Notification (ECN) statistics are supported only on Cisco Nexus 9364C, 9336C-FX, and 93240YC-FX switches.

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 disables the connection to the gRPC receiver if the gRPC receiver sends 20 errors. You will then must unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

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

Telemetry Compression for gRPC Transport

Telemetry compression support is available for gRPC transport. You can use the **use-compression gzip** command to enable compression. (Disable compression with the **no use-compression gzip** command.)

The following example enables compression:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-compression gzip
```

The following example shows that compression is enabled:

```
switch(conf-tm-dest)# show telemetry transport 0 stats

Session Id:                     0
Connection Stats
   Connection Count             0
   Last Connected:              Never
   Disconnect Count             0
   Last Disconnected:           Never
Transmission Stats
   Compression:                 gzip

   Transmit Count:              0
   Last TX time:                None
   Min Tx Time:                 0                    ms
   Max Tx Time:                 0                    ms
   Avg Tx Time:                 0                    ms
   Cur Tx Time:                 0                    ms


switch2(config-if)# show telemetry transport 0 stats

Session Id: 0
Connection Stats
Connection Count 0
Last Connected: Never
Disconnect Count 0
Last Disconnected: Never
Transmission Stats
Compression: disabled
Source Interface: loopback1(1.1.3.4)
Transmit Count: 0
Last TX time: None
Min Tx Time: 0 ms
Max Tx Time: 0 ms
Avg Tx Time: 0 ms
Cur Tx Time: 0 ms
switch2(config-if)#
```

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

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptCompression": {
                      "attributes": {
```

```
            "name": "gzip"
          }
        }
      }
    ]
  }
}
```

NX-API Sensor Path Limitations

NX-API can collect and stream switch information not yet in the DME using **show** commands. However, using the NX-API instead of streaming data from the DME has inherent scale limitations as outlined:

- The switch backend dynamically processes NX-API calls such as **show** commands,
- NX-API spawns several processes that can consume up to a maximum of 20% of the CPU.
- NX-API data translates from the CLI to XML to JSON.

The following is a suggested user flow to help limit excessive NX-API sensor path bandwidth consumption:

1. Check whether the **show** command has NX-API support. You can confirm whether NX-API supports the command from the VSH with the pipe option: `show <command> | json` or `show <command> | json pretty`.

Note Avoid commands that take the switch more than 30 seconds to return JSON output.

2. Refine the **show** command to include any filters or options.
   - Avoid enumerating the same command for individual outputs; i.e., show vlan id 100, show vlan id 101, and so on. 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 that is required for data collection.
3. Configure telemetry with sensor groups that use NX-API as their data sources. Add the **show** commands as sensor paths
4. Configure telemetry with a cadence of 5 times the processing time of the respective **show** command to limit CPU usage.
5. Receive and process the streamed NX-API output as part of the existing DME collection.
6. Telemetry Event is not supported for NX-API Sensor Path.

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 Streaming Telemetry Using the Cisco 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`	Create an SSL or TLS certificate on the server that receives the data, where the *private*.key file is the private key and the *public*.crt is the public key.

	Command or Action	Purpose

	• 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*
	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.

	Command or Action	Purpose

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 action ensures that the telemetry data streams to the same destination IP address in the new VRF.

Step 9	**sensor-group** *sgrp_id*
	Example:
	`switch(config-telemetry)# sensor-group 100`
	`switch(conf-tm-sensor)#`
		Create a sensor group with ID *srgp_id* and enter sensor group configuration mode.
		Currently only numeric ID values are supported. The sensor group defines nodes that will be monitored for telemetry reporting.

Step 10	(Optional) **data-source** *data-source-type* **Example:** `switch(config-telemetry)# data-source NX-API`	Select a data source. Select from either DME or NX-API as the data source. **Note** DME is the default data source.

Step 11	**path** *sensor_path* **depth 0** [**filter-condition** *filter*] [**alias** *path_alias*]
	Example:
	• The following command is applicable for DME, not for NX-API:
	`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`
		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.

	Command or Action	Purpose

	`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`
		• 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**.
		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:**	Create a destination profile for the outgoing data. When the destination group is linked to a subscription, telemetry data is sent to the IP

	Command or Action	Purpose
	• For IPv4: `switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`	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.

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

The following steps describe how to configure a single telemetry DME stream with a ten second cadence with GPB encoding.

```
switch# configure terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(config-tm-dest)# ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
switch(config-tm-dest)# exit
switch(config-telemetry)# sensor group sg1
switch(config-tm-sensor)# data-source DME
switch(config-tm-dest)# path interface depth unbounded query-condition keep-data-type
switch(config-tm-dest)# subscription 1
switch(config-tm-dest)# dst-grp 1
switch(config-tm-dest)# snsr grp 1 sample interval 10000
```

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

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

This example creates a subscription that streams data for `sys/intf` every 5 seconds to destination IP 1.2.3.4 port 50003, and encrypts the stream using GPB encoding that are 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 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 7

Session Id: 7
IP Address:Port 10.197.137.20:37002
Transport: GRPC
Status: Connected
Last Connected: Tue Aug 04 18:05:03.370 IST
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
```

show telemetry transport <session-id> stats

This command displays details of a specific transport session.

```
switch# show telemetry transport 7 stats

Session Id:                     7
Connection Stats
  Connection Count              2
  Last Connected:               Tue Aug 04 18:05:03.370 IST
  Disconnect Count              0
  Last Disconnected:            Never
Transmission Stats
  Compression: disabled
  Source Interface: not set()
  Transmit Count:               13090
  Last TX time:                 Wed Aug 05 23:04:29.420 IST
  Min Tx Time:                  2 ms
  Max Tx Time:                  39704 ms
  Avg Tx Time:                  13 ms
  Cur Tx Time:                  3 ms
```

show telemetry transport <session-id> errors

This command displays detailed error statistics for a specific transport session.

```
switch# show telemetry transport 7 errors

Session Id:                      7
Connection Errors
Connection Error Count:          0
Transmission Errors
Tx Error Count:                  0
Last Tx Error:                   None
Last Tx Return Code:             OK
```

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`.
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 includes 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:**	The **telemetryCertificate** defines the location of the SSL/TLS certificate with the telemetry subscription/destination.

	Command or Action	Purpose

	<pre>{
	    "telemetryCertificate": {
	        "attributes": {
	            "filename": "root.pem"
	             "hostname": "c.com"
	         }
	    }
	}</pre>	

Step 5	Define a telemetry destination group.
	Example:
	<pre>{
	      "telemetryDestGroup": {
	        "attributes": {
	          "id": "20"
	        }
	      }
	}</pre>
		A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.

Step 6	Define a telemetry destination profile.
	Example:
	<pre>{
	   "telemetryDestProfile": {
	      "attributes": {
	         "adminSt": "enabled"
	      },
	      "children": [
	         {
	
	"telemetryDestOptSourceInterface": {
	                "attributes": {
	                   "name": "lo0"
	                }
	             }
	         }
	      ]
	   }
	}</pre>
		A telemetry destination profile is defined in **telemetryDestProfile**.
		• Configure the **adminSt** attribute as `enabled`.
		• Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream data from the configured interface to a destination with the source IP address.

Step 7	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
	Example:
	<pre>{
	    "telemetryDest": {
	        "attributes": {
	             "addr": "1.2.3.4",
	             "enc": "GPB",
	             "port": "50001",
	             "proto": "gRPC",
	             "rn":
	"addr-[1.2.3.4]-port-50001"
	         }
	    }</pre>
		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.

	Command or Action	Purpose
	<pre>}</pre>	• 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 • UDP and secure UDP (DTLS)

Step 8	Create a telemetry subscription to configure the telemetry behavior.
	Example:
	<pre>"telemetrySubscription": {
	    "attributes": {
	        "id": "30",
	        "rn": "subs-30"
	    }, "children": [{
	    }]
	}</pre>
		A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object:
		• **id** — An identifier for the subscription. Currently only numeric ID values are supported.
		• **rn** — The relative name of the subscription object in the format: **subs-***id*.
		Children of the subscription object includes 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:
	<pre>{
	      "telemetrySubscription": {
	        "attributes": {
	          "id": "30"
	        }
	        "children": [{
	          "telemetryRsSensorGroupRel":
	{
	            "attributes": {
	              "sampleIntvl": "5000",
	              "tDn": "sys/tm/sensor-10"
	
	            }
	          }
	        }
	        ]
	      }
	    }</pre>	

	Command or Action	Purpose
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**. • **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. **Example:** Single sensor path `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" }`	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

	Command or Action	Purpose

	<pre> }
	}</pre>
	Example:
	Single sensor path for NX-API
	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "show interface",
	            "path": "show bgp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>
	Example:
	Multiple sensor paths
	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	},
	{
	     "telemetrySensorPath": {
	         "attributes": {
	            "excludeFilter": "",
	            "filterCondition": "",
	            "path": "sys/fm/dhcp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>
	Example:
	Single sensor path filtering for BGP disable events:
		about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635

	Command or Action	Purpose

	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition":
	"eq(fmBgp.operSt.\"disabled\")",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>	

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**).	

Step 15	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
	Example:
	<pre>"telemetryRtSensorGroupRel": {
	    "attributes": {
	        "rn":
	"rtsensorGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		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:
	<pre>"telemetryRtDestGroupRel": {
	    "attributes": {
	        "rn":
	"rtdestGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	
	        "tDn": "sys/tm/subs-30"</pre>
		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**.

	Command or Action	Purpose
	<pre> } }</pre>	• **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:
	<pre>"telemetryRsDestGroupRel": {
	    "attributes": {
	        "rType": "mo",
	        "rn":
	"rsdestGroupRel-[sys/tm/dest-20]",
	        "tCl": "telemetryDestGroup",
	        "tDn": "sys/tm/dest-20",
	        "tType": "mo"
	    }
	}</pre>
		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.
		• **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 that are collected into one POST payload (note that some attributes may not match):

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
```

```
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50051",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "5000",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

Configuration Example for Telemetry Using the NX-API

Streaming Paths to a Destination

This example creates a subscription that streams paths `sys/cdp` and `sys/ipv4` to a destination `1.2.3.4 port 50001` every five seconds.

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
```

```
"telemetryEntity": {
    "attributes": {
        "dn": "sys/tm"
    },
    "children": [{
        "telemetrySensorGroup": {
            "attributes": {
                "id": "10",
                "rn": "sensor-10"
            },  "children": [{
                "telemetryRtSensorGroupRel": {
                    "attributes": {
                        "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/cdp",
                        "rn": "path-[sys/cdp]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/ipv4",
                        "rn": "path-[sys/ipv4]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }]
        }
    }, {
        "telemetryDestGroup": {
            "attributes": {
                "id": "20",
                "rn": "dest-20"
            },
            "children": [{
                "telemetryRtDestGroupRel": {
                    "attributes": {
                        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetryDest": {
                    "attributes": {
                        "addr": "1.2.3.4",
                        "enc": "GPB",
                        "port": "50001",
                        "proto": "gRPC",
```

```
                            "rn": "addr-[1.2.3.4]-port-50001"
                        }
                    }
                }]
            }
        }, {
            "telemetrySubscription": {
                "attributes": {
                    "id": "30",
                    "rn": "subs-30"
                },
                "children": [{
                    "telemetryRsDestGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                            "tCl": "telemetryDestGroup",
                            "tDn": "sys/tm/dest-20",
                            "tType": "mo"
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

Filter Conditions on BGP Notifications

The following example payload enables notifications that trigger when the BFP feature is disabled as per the `filterCondition` attribute in the `telemetrySensorPath` MO. The data is streamed to`10.30.217.80 port 50055`.

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
```

```
            "depth": "0"
          }
        }
      }
      ]
    }
  },
  {
    "telemetryDestGroup": {
      "attributes": {
        "id": "20"
      }
      "children": [{
        "telemetryDest": {
          "attributes": {
            "addr": "10.30.217.80",
            "port": "50055",
            "enc": "GPB",
            "proto": "gRPC"
          }
        }
      }
      ]
    }
  },
  {
    "telemetrySubscription": {
      "attributes": {
        "id": "30"
      }
      "children": [{
        "telemetryRsSensorGroupRel": {
          "attributes": {
            "sampleIntvl": "0",
            "tDn": "sys/tm/sensor-10"
          }
        }
      },
      {
        "telemetryRsDestGroupRel": {
          "attributes": {
            "tDn": "sys/tm/dest-20"
          }
        }
      }
      ]
    }
  }
  ]
 }
}
```

Using Postman Collection for Telemetry Configuration

An example Postman collection is an easy way to start configuring the telemetry feature, and can run all telemetry CLI equivalents in a single payload. Modify the file in the preceding link using your preferred text editor to update the payload to your needs, then open the collection in Postman and run the collection.

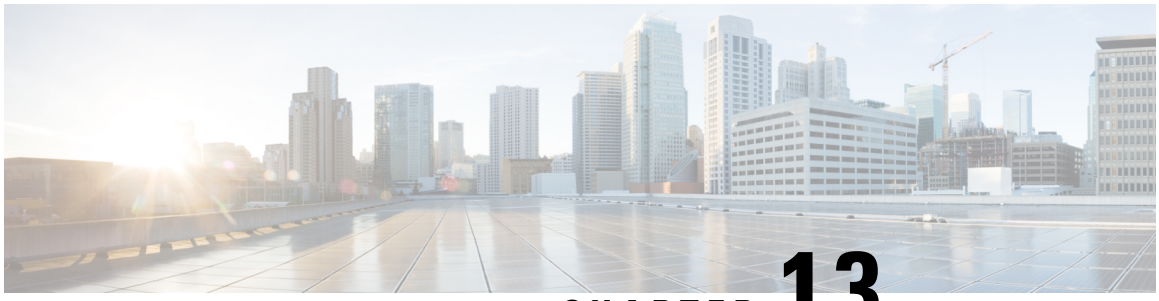
Telemetry Model in the DME

The telemetry application is modeled in the DME with the following structure:

```
model
|----package [name:telemetry]
    |  @name:telemetry
    |----objects
         |----mo [name:Entity]
              |    @name:Entity
              |      @label:Telemetry System
              |--property
              |    @name:adminSt
              |      @type:AdminState
              |
              |----mo [name:SensorGroup]
              |    |    @name:SensorGroup
              |    |      @label:Sensor Group
              |    |--property
              |    |    @name:id [key]
              |    |      @type:string:Basic
              |    |    @name:dataSrc
              |    |      @type:DataSource
              |    |
              |    |----mo [name:SensorPath]
              |         |   @name:SensorPath
              |         |     @label:Sensor Path
              |         |--property
              |         |    @name:path [key]
              |         |      @type:string:Basic
              |         |    @name:filterCondition
              |         |      @type:string:Basic
              |         |    @name:excludeFilter
              |         |      @type:string:Basic
              |         |    @name:depth
              |         |      @type:RetrieveDepth
              |
              |----mo [name:DestGroup]
              |    |    @name:DestGroup
              |    |      @label:Destination Group
              |    |--property
              |    |    @name:id
              |    |      @type:string:Basic
              |    |
              |    |----mo [name:Dest]
              |         |   @name:Dest
              |         |     @label:Destination
              |         |--property
              |         |    @name:addr [key]
              |         |      @type:address:Ip
              |         |    @name:port [key]
              |         |      @type:scalar:Uint16
              |         |    @name:proto
              |         |      @type:Protocol
              |         |    @name:enc
              |         |      @type:Encoding
              |
              |----mo [name:Subscription]
                   |    @name:Subscription
                   |      @label:Subscription
                   |--property
                   |    @name:id
                   |      @type:scalar:Uint64
                   |----reldef
                   |    |  @name:SensorGroupRel
                   |    |    @to:SensorGroup
                   |    |    @cardinality:ntom
```

```
|     |    @label:Link to sensorGroup entry
|     |--property
|          @name:sampleIntvl
|            @type:scalar:Uint64
|
|----reldef
      |  @name:DestGroupRel
      |    @to:DestGroup
      |    @cardinality:ntom
      |    @label:Link to destGroup entry
```

DNs Available to Telemetry



CHAPTER 13

XML Management Interface

This section contains the following topics:

- About the XML Management Interface, on page 125
- Licensing Requirements for the XML Management Interface, on page 126
- Prerequisites to Using the XML Management Interface, on page 127
- Using the XML Management Interface, on page 127
- Information About Example XML Instances, on page 139
- Additional References, on page 145

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 129 and RFC 4741.

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

This section includes the following topics:

- NETCONF Layers, on page 125
- SSH xmlagent, on page 126

NETCONF Layers

The following are the NETCONF layers:

Table 8: 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 129 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 9: NETCONF Operations in Cisco NX-OS

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	NETCONF Close Session Instance, on page 139
commit	Sets the running configuration to the current contents of the candidate configuration.	NETCONF Commit Instance - Candidate Configuration Capability, on page 144
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 144
copy-config	Copies the content of source configuration datastore to the target datastore.	NETCONF copy-config Instance, on page 140
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 140 NETCONF rollback-on-error Instance , on page 144
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 129
get-config	Retrieves all or part of a configuration	NETCONF get-config Instance, on page 142
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 140

NETCONF Operation	Description	Example
lock	Allows the client to lock the configuration system of a device.	NETCONF Lock Instance, on page 142
unlock	Releases the configuration lock that the session issued.	NETCONF unlock Instance, on page 143
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	NETCONF validate Capability Instance , on page 145

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 128 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 129 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 138 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 10: Tags

Tag	Description
<exec-command>	Executes a CLI command.

Tag	Description
<cmd>	Contains the CLI command. A command can be a show or configuration command. Separate multiple configuration commands by using a semicolon ";". Multiple show commands are not supported. You can send multiple configuration commands in different <cmd> tags as part of the same request. For more information, see the Example in *Configuration CLI Commands Sent Through <exec-command>*.

Replies to configuration commands that are sent through the <cmd> tag are as follows:

- <nf:ok>: All configure commands are executed successfully.
- <nf:rpc-error>: Some commands have failed. The operation stops on the first error, and the <nf:rpc-error> subtree provides more information on what configuration failed. Notice that any configuration that is executed before the failed command would have been applied to the running configuration.

The following example shows a failed configuration:

Failed Configuration

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 1.1.1.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

Because of a command execution, the interface IP address is set, but the administrative state is not modified (the no shut command is not executed). The reason the administrative state is not modified is because the no port-channel 2000 command results in an error.

The <rpc-reply> results from a show command that is sent through the <cmd> tag that contains the XML output of the show command.

You cannot combine configuration and show commands on the same <exec-command> instance. The following example shows a configuration and **show** command that are combined in the same instance.

Combination of Configuration and show Commands

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

The show command must be sent in its own <exec-command> instance as shown in the following example:

Show CLI Commands Sent Through <exec-command>

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

NETCONF Replies

For every XML request sent by the client, the XML server sends an XML response enclosed in the RPC response tag <rpc-reply>.

This section contains the following topics:

• RPC Response Tag, on page 138
• Interpreting Tags Encapsulated in the Data Tag, on page 138

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 11: 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 139
- NETCONF Kill-session Instance, on page 140
- NETCONF copy-config Instance, on page 140
- NETCONF edit-config Instance, on page 140
- NETCONF get-config Instance, on page 142
- NETCONF Lock Instance, on page 142
- NETCONF unlock Instance, on page 143
- NETCONF Commit Instance - Candidate Configuration Capability, on page 144
- NETCONF Confirmed-commit Instance , on page 144
- NETCONF rollback-on-error Instance , on page 144
- NETCONF validate Capability Instance , on page 145

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

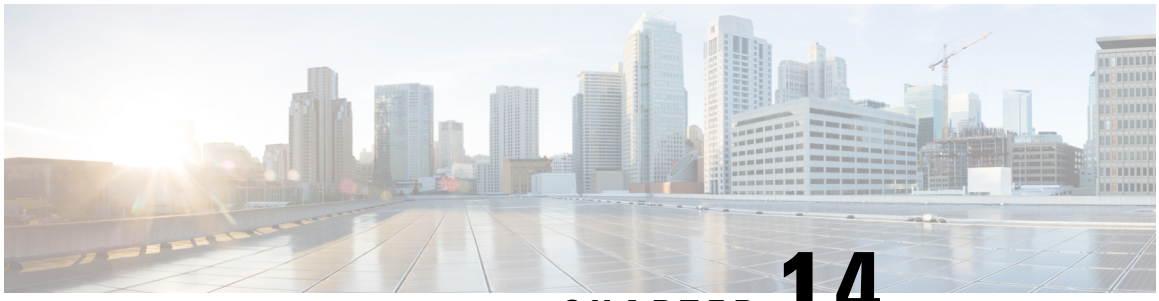
This section provides additional information that is related to implementing the XML management interface.

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)



CHAPTER 14

Converting CLI Commands to Network Configuration Format

- Information About XMLIN, on page 147
- Licensing Requirements for XMLIN, on page 147
- Installing and Using the XMLIN Tool, on page 148
- Converting Show Command Output to XML, on page 148
- Configuration Examples for XMLIN, on page 149

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 12: 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>
]]>]]>
```