# NX-API CLI



# About NX-API CLI

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

NX-API CLI supports JSON-RPC.

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

## Transport

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

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

**Note** For the 7.x release, the Nginx process continues to run even after NX-API is disabled using the “no feature NXAPI” command. This is required for other management-related processes. In the 6.x release, all processes were killed when you ran the “no feature NXAPI” command, so this is a change in behavior in the 7.x release.

# Message Format

**Note**

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

# Security

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

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

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

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

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

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

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

# Using NX-API CLI

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

**Note** For more details about NX-API response codes, see Table of NX-API Response Codes, on page 20.

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

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

- Enable the management interface.

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

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

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

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

Request:

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

Response:

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

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

Request:

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

```
}
```

Response:

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

# Escalate Privileges to Root on NX-API

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

The following are guidelines for escalating privileges:

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

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

First example:

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

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

```
</ins_api>
```

Second example:

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

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

# NX-API Management Commands

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

***Table 1: NX-API Management Commands***

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

| NX-API Management Command | Description |
|---|---|
| **nxapi certificate {httpscrt certfile** \| **httpskey keyfile**} *filename* | Specifies the upload of the following:<br>• HTTPS certificate when httpscrt is specified.<br>• HTTPS key when httpskey is specified.<br>Example of HTTPS certificate:<br>`nxapi certificate httpscrt certfile bootflash:cert.crt`<br>Example of HTTPS key:<br>`nxapi certificate httpskey keyfile bootflash:privkey.key` |
| **nxapi certificatehttpskey keyfile** *filename* **password** *passphrase* | Installs NX-API certificates with encrypted private keys:<br>**Note** The passphrase for decrypting the encrypted private key is **pass123!**.<br>Example:<br>`nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!` |
| **nxapi certificate enable** | Enables a certificate. |

| NX-API Management Command | Description |
|---|---|
| **nxapi certificate trustpoint <trustpoint label>** | Beginning with Cisco NX-OS release 10.2(3)F, the user can now import the certificate or use the CA certificate for the NX-API using the trustpoint infra.<br>**Note** Refer to the *Cisco Nexus 9000 Security Configuration Guide* to configure the crypto ca import trustpoint to first import certificate.<br>**Note** Currently only pkcs12 certificate import is supported in this form. The NX-API certificate enable/NX-API certificate trustpoint and NX-API certificate sudi are mutually exclusive and each configuration will overwrite the certificate/key.<br>**Note** The maximum size of cert/key supported with NX-API certificate enable is 8k. If the size is >8k, use NX-API certificate trustpoint to import the certificate.<br>**Note** If you have configured a custom certificate in NX-API using trustpoint infra, upon entering the **reload ascii** command the configuration is lost. It will revert to the default day-1 NX-API certificate. After entering the **reload ascii** command, the switch will reload. Once the switch is up again, you need to reconfigure the NX-API certificate trustpoint configuration.<br>**Note** Beginning with Cisco NX-OS Release 10.3(1)F, support for ascii truspoint reload is added.<br>**Note** Config-replace will fail if the current running-config do not contain the trustpoint and certificate imported, but the target config contains the creation of trustpoint "crypto ca trustpoint <trustpoint name>" and "nxapi certificate trustpoint <trustpoint-name>" CLI. If trustpoint is not present, then first you need to create trustpoint and import certificate before attempting "nxapi certificate trustpoint <trustpoint-label>". |

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

| NX-API Management Command | Description |
|---|---|
| **nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2 TLSv1.3}** | Beginning with Cisco NX-OS Release 10.2(4)M, TLSv1.3 is supported on Cisco Nexus 9000 series platform switches. Running this command enables the TLS versions specified in the string. Beginning with Cisco NX-OS Release 9.3(2), only TLSv1.2 is enabled by default.<br>The **no** form of the command changes the TLS version to the default version.<br>• If you want to enable particular TLS version, specify only that respective TLS version.<br>For example, if you need TLSv1.3, use the following command:<br>`switch(config)# nxapi ssl protocols TLSv1.3`<br>• If you want to enable multiple TLS versions for backward compatibility at a later stage, specify all the required TLS versions that are supported.<br>For example:<br>• If you need TLSv1.1 through TLSv1.3, use the following command to enable all required TLS versions:<br>`switch(config)# nxapi ssl protocols TLSv1.2 TLSv1.3`<br>• When you need backward compatibility, use the following command to enable that version:<br>`switch(config)# nxapi ssl protocols TLSv1.2`<br>**Note** • It is recommended to use TLSv1.2 and TLSv1.3 for backward compatibility.<br>`switch(config)# nxapi ssl protocols TLSv1.2 TLSv1.3`<br>For example, if you are :<br>• Before configuring TLSv1.3, validate the server and client certificates for TLSv1.3 support.<br>• NX-API server side SUDI certificate is not supported with TLSv1.3. |
| **nxapi use-vrf** *vrf* | Specifies the default VRF, management VRF, or named VRF.<br>**Note** In Cisco NX-OS Release 7.0(3)I2(1) NGINX listens on only one VRF. |

| NX-API Management Command | Description |
|---|---|
| **system server session cmd-timeout <timeout>** | Beginning with Cisco NX-OS release, 10.2(3)F, in NGINX server, the default timeout to run any command is 5 minutes. The users can increase the timeout to the desired value from 60 seconds (1 minute) to 3600 seconds (1 hour) according to their need and time taken for executing the commands. |
| **ip netns exec management iptables** | Implements any access restrictions and can be run in management VRF.<br>**Note** You must enable **feature bash-shell** and then run the command from Bash Shell. For more information on Bash Shell, see the chapter on Bash.<br>*Iptables is a command-line firewall utility that uses policy chains to allow or block traffic and almost always comes pre-installed on any Linux distribution.*<br>**Note** For more information about making iptables persistent across reloads when they are modified in a bash-shell, see Making an Iptable Persistent Across Reloads, on page 19. |
| **nxapi idle-timeout <timeout>** | Starting with Release 9.3(5), you can configure the amount of time before an idle NX-API session is invalidated. The time can be 1 - 1440 minutes. The default time is 10 minutes. Return to the default value by using the no form of the command: **no nxapi idle-timeout <timeout>** |

The following is an example for NX-API output for SUDI:

```
switch(config)# nxapi certificate sudi
switch# show nxapi
nxapi enabled
NXAPI timeout 10
NXAPI cmd timeout 300
HTTP Listen on port 80
HTTPS Listen on port 443
Certificate Information:
    Issuer:   issuer=CN = High Assurance SUDI CA, O = Cisco
    Expires:  Aug  9 20:58:26 2099 GMT
switch#
switch#
switch# show run | sec nxapi
feature nxapi
nxapi http port 80
nxapi certificate sudi
switch#
```

The following is an example for trustpoint configuration:

```
switch(config)# crypto ca trustpoint ngx
switch(config-trustpoint)# crypto ca import ngx pkcs12 bootflash:server.pfx cisco123
witch(config)# nxapi certificate trustpoint ngx
switch(config)# show nxapi
nxapi enabled
NXAPI timeout 10
NXAPI cmd timeout 300
HTTP Listen on port 80
```

```
Trustpoint label ngx
HTTPS Listen on port 443
Certificate Information:
Issuer: issuer=C = IN, ST = KA, L = bang, O = cisco, OU = nxpi, CN = suprss@cisco.com,
emailAddress = suprss@cisco.com
Expires: Jan 13 06:13:50 2023 GMT
switch(config)#
switch(config)# show run | sec nxapi
feature nxapi
nxapi http port 80
nxapi certificate trustpoint ngx
```

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

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

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

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

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

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

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

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

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

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

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

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

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

# Working With Interactive Commands Using NX-API

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

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

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

# NX-API Request Elements

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

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

***Table 2: NX-API Request Elements for XML or JSON Format***

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

| NX-API Request Element | Description |
| --- | --- |
| *type* | Specifies the type of command to be executed.<br>The following types of commands are supported:<br>• **cli_show**<br>CLI **show** commands that expect structured output. If the command does not support XML output, an error message is returned.<br>• **cli_show_array**<br>CLI **show** commands that expect structured output. Only for show commands. Similar to **cli_show**, but with **cli_show_array**, data is returned as a list of one element, or an array, within square brackets [ ].<br>• **cli_show_ascii**<br>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.<br>• **cli_conf**<br>CLI configuration commands.<br>• **bash**<br>Bash commands. Most non-interactive Bash commands are supported by NX-API.<br>**Note**<br>• Each command is only executable with the current user's authority.<br>• 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.<br>• A maximum of 10 consecutive **show** commands are supported. If the number of **show** commands exceeds 10, the 11th and subsequent commands are ignored.<br>• 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.<br><br>Enable or disable chunk with the following settings:<br><br>
<table>
<tr><td>0</td><td>Do not chunk output.</td></tr>
<tr><td>1</td><td>Chunk output.</td></tr>
</table>
Note<br>• Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.<br>• The output message format options are XML or JSON.<br>• For the XML output message format , special characters, such as < or >, are converted to form a valid XML message (< is converted into < > is converted into &gt).<br>You can use XML SAX to parse the chunked output.<br>• When the output message format is JSON, the chunks are concatenated to create a valid JSON object.<br><br>Note When chunking is enabled, the maximum message size supported is currently 200MB of chunked output.</td></tr>
<tr><td>rollback</td><td>Valid only for configuration CLIs, not for show commands. Specifies the configuration rollback options. Specify one of the following options.<br>• Stop-on-error—Stops at the first CLI that fails.<br>• Continue-on-error—Ignores and continues with other CLIs.<br>• Rollback-on-error—Performs a rollback to the previous state the system configuration was in.<br><br>Note The rollback element is available in the cli_conf mode when the input request format is XML or JSON.</td></tr>
</table>

<table>
<tr><th>NX-API Request Element</th><th>Description</th></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.<br><br>NX-OS release 9.3(1) introduces the sid option <code>clear</code>. When a new chunk request is initiated with the sid set to <code>clear</code>, all current chunk requests are discarded or abandoned.<br><br>When you receive response code <code>429: Max number of concurrent chunk request is 2</code>, use sid <code>clear</code> to abandon the current chunk requests. After using sid <code>clear</code>, subsequent response codes operate as usual per the rest of the request.</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.<br><br>Note Except for bash, multiple commands are separated with " ; ". (The ; must be surrounded with single blank characters.)<br><br>For bash, multiple commands are separated with ";". (The ; is not surrounded with single blank characters.)<br><br>The following are examples of multiple commands:<br><br><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>
<tr><td>output_format</td><td>The available output message formats are the following:<br><br><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></td></tr>
</table>

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

***Table 3: NX-API Request Elements for JSON-RPC Format***

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

| NX-API Request Element | Description |
|---|---|
| *method* | A string containing the name of the method to be invoked.<br>NX-API supports either:<br>• **cli**−show or configuration commands<br>• **cli_ascii**−show or configuration commands; output without formatting<br>• **cli_array**−only for show commands; similar to **cli**, but with **cli_array**, data is returned as a list of one element, or an array, within square brackets, [ ]. |
| *params* | A structured value that holds the parameter values used during the invocation of a method.<br>It must contain the following:<br>• **cmd**−CLI command<br>• **version**−NX-API request version identifier |
| *rollback* | Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options.<br>• Stop-on-error—Stops at the first CLI that fails.<br>• Continue-on-error—Ignores the failed CLI and continues with other CLIs.<br>• Rollback-on-error—Performs a rollback to the previous state the system configuration was in. |
| *validate* | Configuration validation settings. This element allows you to validate the commands before you apply them on the switch. This enables you to verify the consistency of a configuration (for example, the availability of necessary hardware resources) before applying it. Choose the validation type from the Validation Type drop-down list.<br>• **Validate-Only**—Validates the configurations, but does not apply the configurations.<br>• **Validate-and-Set**—Validates the configurations, and applies the configurations on the switch if the validation is successful. |
| *lock* | An exclusive lock on the configuration can be specified, whereby no other management or programming agent will be able to modify the configuration if this lock is held. |

| NX-API Request Element | Description |
|---|---|
| *id* | An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the id parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, *id* : 1 |

# NX-API Response Elements

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

***Table 4: 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.<br>When multiple commands are in cli_show or cli_show_ascii, each command output is enclosed by a single output tag.<br>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.<br>For cli_conf and bash message types, this element contains the outputs of all the commands. |
| input | Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element. |
| body | Body of the command response. |
| code | Error code returned from the command execution.<br>NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml). |
| msg | Error message associated with the returned error code. |

# Restricting Access to NX-API

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

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

If you have configured a VRF for NX-API communication, however, ACLs will not restrict HTTP or HTTPS access. Instead, create a rule for an iptable. For more information about creating a rule, see Updating an iptable, on page 18.

## Updating an iptable

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

**Step 1** To create a rule that blocks HTTP access:

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

**Step 2** To create a rule that blocks HTTPS access:

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

**Step 3** To verify the applied rules:

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


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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

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

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


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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

**Step 5** To remove and verify previously applied rules:

This example removes the first rule from INPUT.

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

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

### What to do next

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. To make the rules persistent, see Making an Iptable Persistent Across Reloads, on page 19.

## Making an Iptable Persistent Across Reloads

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

### Before you begin

You have modified an iptable.

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

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

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

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

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

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:

# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables
```

```
# Description:       sets config for iptables

#                    during boot time

### END INIT INFO


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
  start)
    start_script
   ;;
  stop)
    ;;
  restart)
    sleep 1
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac
exit 0
```

**Step 4** Set the appropriate permissions to the startup script:

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

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

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

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

# Table of NX-API Response Codes

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

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

***Table 5: NX-API Response Codes***

| NX-API Response | Code | Message |
|---|---|---|
| SUCCESS | 200 | Success. |
| CUST_OUTPUT_PIPED | 204 | Output is piped elsewhere due to request. |

| | | |
|---|---|---|
| BASH_CMD_ERR | 400 | Input Bash command error. |
| CHUNK_ALLOW_ONE_CMD_ERR | 400 | Chunking only allowed to one command. |
| CLI_CLIENT_ERR | 400 | CLI execution error. |
| CLI_CMD_ERR | 400 | Input CLI command error. |
| EOC_NOT_ALLOWED_ERR | 400 | The `eoc` value is not allowed as session Id in the request. |
| IN_MSG_ERR | 400 | Request message is invalid. |
| MSG_VER_MISMATCH | 400 | Message version mismatch. |
| NO_INPUT_CMD_ERR | 400 | No input command. |
| SID_NOT_ALLOWED_ERR | 400 | Invalid character that is entered as a session ID. |
| PERM_DENY_ERR | 401 | Permission denied. |
| CONF_NOT_ALLOW_SHOW_ERR | 405 | Configuration mode does not allow **show** . |
| SHOW_NOT_ALLOW_CONF_ERR | 405 | Show mode does not allow configuration. |
| EXCEED_MAX_SHOW_ERR | 413 | Maximum number of consecutive show commands exceeded. The maximum is 10. |
| MSG_SIZE_LARGE_ERR | 413 | Response size too large. |
| RESP_SIZE_LARGE_ERR | 413 | Response size stopped processing because it exceeded the maximum message size. The maximum is 200 MB. |
| EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR | 429 | Maximum number of concurrent chunk requests is exceeded. The maximum is 2. |
| OBJ_NOT_EXIST | 432 | Requested object does not exist. |
| BACKEND_ERR | 500 | Backend processing error. |
| CREATE_CHECKPOINT_ERR | 500 | Error creating a checkpoint. |
| DELETE_CHECKPOINT_ERR | 500 | Error deleting a checkpoint. |
| FILE_OPER_ERR | 500 | System internal file operation error. |
| LIBXML_NS_ERR | 500 | System internal LIBXML NS error. |
| LIBXML_PARSE_ERR | 500 | System internal LIBXML parse error. |
| LIBXML_PATH_CTX_ERR | 500 | System internal LIBXML path context error. |
| MEM_ALLOC_ERR | 500 | System internal memory allocation error. |
| ROLLBACK_ERR | 500 | Error executing a rollback. |

| | | |
|---|---|---|
| SERVER_BUSY_ERR | 500 | Request is rejected because the server is busy. |
| USER_NOT_FOUND_ERR | 500 | User not found from input or cache. |
| VOLATILE_FULL | 500 | Volatile memory is full. Free up memory space and retry. |
| XML_TO_JSON_CONVERT_ERR | 500 | XML to JSON conversion error. |
| BASH_CMD_NOT_SUPPORTED_ERR | 501 | Bash command not supported. |
| CHUNK_ALLOW_XML_ONLY_ERR | 501 | Chunking allows only XML output. |
| CHUNK_ONLY_ALLOWED_IN_SHOW_ERR | 501 | Response chunking allowed only in `show` commands. |
| CHUNK_TIMEOUT | 501 | Timeout while generating chunk response. |
| CLI_CMD_NOT_SUPPORTED_ERR | 501 | CLI command not supported. |
| JSON_NOT_SUPPORTED_ERR | 501 | JSON not supported due to large amount of output. |
| MALFORMED_XML | 501 | Malformed XML output. |
| MSG_TYPE_UNSUPPORTED_ERR | 501 | Message type not supported. |
| OUTPUT_REDIRECT_NOT_SUPPORTED_ERR | 501 | Output redirection is not supported. |
| PIPE_OUTPUT_NOT_SUPPORTED_ERR | 501 | Pipe operation not supported. |
| PIPE_XML_NOT_ALLOWED_IN_INPUT | 501 | Pipe XML is not allowed in input. |
| PIPE_NOT_ALLOWED_IN_INPUT | 501 | Pipe is not allowed for this input type. |
| RESP_BIG_USE_CHUNK_ERR | 501 | Response is greater than the allowed maximum. The maximum is 10 MB. Use XML or JSON output with chunking enabled. |
| RESP_BIG_JSON_NOT_ALLOWED_ERR | 501 | Response has large amount of output. JSON not supported. |
| STRUCT_NOT_SUPPORTED_ERR | 501 | Structured output unsupported. |
| ERR_UNDEFINED | 600 | Undefined. |

# XML and JSON Supported Commands

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

- XML
- JSON

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

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

Selected examples of this feature follow.

# About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard designed for human-readable data and is an alternative to XML. JSON was originally designed from JavaScript, but it is language-independent data format. JSON Pretty format, as well as JSON Native and JSON Pretty Native, is also supported.

The JSON/CLI Execution is currently supported in N3500.

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

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

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

CLI Execution

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

# Examples of XML and JSON Output

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

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096",
 "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "
used_mcast_oifl": "2", "used_host_in_host_total": "13", "used_host4_in_host": "1
2", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table":
"0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"
}
```

```
switch(config)#
```

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

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

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

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

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

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

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

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

```
        </__readonly__>
       </__XML__OPT_Cmd_show_acl___readonly__>
      </__XML__OPT_Cmd_show_acl_expanded>
     </__XML__OPT_Cmd_show_acl_capture>
    </__XML__OPT_Cmd_show_acl_name>
   </access-lists>
  </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
 </show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#
```

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

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

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

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

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

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

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

```
switch-1# show ip route summary | xml
<?xml version="1.0" encoding="ISO-8859-1"?> <nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:urib" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">

 <nf:data>
  <show>
   <ip>
    <route>
     <__XML__OPT_Cmd_urib_show_ip_route_command_ip>
      <__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
       <__XML__OPT_Cmd_urib_show_ip_route_command_topology>
        <__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
         <__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
          <__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
           <__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
            <__XML__OPT_Cmd_urib_show_ip_route_command_summary>
             <__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
```

```
<__XML__OPT_Cmd_urib_show_ip_route_command___readonly__>
 <__readonly__>
  <TABLE_vrf>
   <ROW_vrf>
    <vrf-name-out>default</vrf-name-out>
    <TABLE_addrf>
     <ROW_addrf>
      <addrf>ipv4</addrf>
      <TABLE_summary>
       <ROW_summary>
        <routes>938</routes>
        <paths>1453</paths>
        <TABLE_unicast>
         <ROW_unicast>
          <clientnameuni>am</clientnameuni>
          <best-paths>2</best-paths>
         </ROW_unicast>
         <ROW_unicast>
          <clientnameuni>local</clientnameuni>
          <best-paths>105</best-paths>
         </ROW_unicast>
         <ROW_unicast>
          <clientnameuni>direct</clientnameuni>
          <best-paths>105</best-paths>
         </ROW_unicast>
         <ROW_unicast>
          <clientnameuni>broadcast</clientnameuni>
          <best-paths>203</best-paths>
         </ROW_unicast>
         <ROW_unicast>
          <clientnameuni>ospf-10</clientnameuni>
          <best-paths>1038</best-paths>
         </ROW_unicast>
        </TABLE_unicast>
        <TABLE_route_count>
         <ROW_route_count>
          <mask_len>8</mask_len>
          <count>1</count>
         </ROW_route_count>
         <ROW_route_count>
          <mask_len>24</mask_len>
          <count>600</count>
         </ROW_route_count>
         <ROW_route_count>
          <mask_len>31</mask_len>
          <count>13</count>
         </ROW_route_count>
         <ROW_route_count>
          <mask_len>32</mask_len>
          <count>324</count>
         </ROW_route_count>
        </TABLE_route_count>
       </ROW_summary>
      </TABLE_summary>
     </ROW_addrf>
    </TABLE_addrf>
   </ROW_vrf>
  </TABLE_vrf>
 </__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command___readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
</__XML__OPT_Cmd_urib_show_ip_route_command_summary>
</__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
```

```
        </__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
       </__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
      </__XML__OPT_Cmd_urib_show_ip_route_command_topology>
     </__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
    </__XML__OPT_Cmd_urib_show_ip_route_command_ip>
   </route>
  </ip>
 </show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1#
```

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

```
switch-1# show ip ospf | json native
{"TABLE_ctx":{"ROW_ctx":[{"ptag":"Blah","instance_number":4,"cname":"default","
rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr
_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_t
os0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false"
,"admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S"
,"spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_
time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"m
ax_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cn
t":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_
nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa
":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","ins
tance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":
"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive"
,"gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true"
,"is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_
time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0
S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_
aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"
asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"
area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal"
:0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_
rt_int":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0
.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_pe
riod":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only"
:"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_d
ist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max
_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT
5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric
_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"aso
paque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"
act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_d
iscard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_num
ber":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","g
r_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last
_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr"
:"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT
0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_h
old_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pac
e":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa
_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_norm
al":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_a
rea_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"
false"}]}}
switch-1#
```

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

```
switch-1# show ip ospf | json-pretty native
{
```

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

                        ... content deleted for brevity ...

                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
                        "area_stub":    0,
                        "area_nssa":    0,
```

```
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                }]
        }
}
switch-1#
```

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

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

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

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

```
                                        "mask_len": "8",
                                        "count": "1"
                                    },
                                    {
                                        "mask_len": "24",
                                        "count": "600"
                                    },
                                    {
                                        "mask_len": "31",
                                        "count": "13"
                                    },
                                    {
                                        "mask_len": "32",
                                        "count": "324"
                                    }
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
}
switch-1#
```