



NX-API

- [About NX-API, on page 1](#)
- [Using NX-API, on page 2](#)
- [XML and JSON Supported Commands, on page 10](#)

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 {https crt |https key}
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.

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 management
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 1: NX-API Management Commands

NX-API Management Command	Description
feature nxapi	Enables NX-API.

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

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

```
switch(config)# nxapi certificate httpsert 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
Ngix 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](#) for more information on 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 2: NX-API Request Elements

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

NX-API Request Element	Description
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> • 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. <p>Note</p> <ul style="list-style-type: none"> • Each command is only executable with the current user's authority. • The pipe operation is supported in the output when the message type is ASCII. If the output is in XML format, the pipe operation is not supported. • A maximum of 10 consecutive show commands are supported. If the number of show commands exceeds 10, the 11th and subsequent commands are ignored. • No interactive commands are supported.

NX-API Request Element	Description						
<i>chunk</i>	<p>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.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="786 485 1481 596"> <tr> <td data-bbox="786 485 899 541">0</td> <td data-bbox="899 485 1481 541">Do not chunk output.</td> </tr> <tr> <td data-bbox="786 541 899 596">1</td> <td data-bbox="899 541 1481 596">Chunk output.</td> </tr> </table> <p>Note Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.</p> <p>The output message format is XML. (XML is the default.) Special characters, such as < or >, are converted to form a valid XML message (< is converted into &lt; > is converted into &gt;).</p> <p>You can use XML SAX to parse the chunked output.</p> <p>Note When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</p>	0	Do not chunk output.	1	Chunk output.		
0	Do not chunk output.						
1	Chunk output.						
<i>sid</i>	<p>The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a <i>sid</i> to match the <i>sid</i> of the previous response message.</p>						
<i>input</i>	<p>Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, show commands are cli_show message type and are not supported in cli_conf mode.</p> <p>Note Except for bash, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.)</p> <p>For bash, multiple commands are separated with ";". (The ; is not surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="786 1587 1481 1814"> <tr> <td data-bbox="786 1587 911 1665">cli_show</td> <td data-bbox="911 1587 1481 1665">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="786 1665 911 1740">cli_conf</td> <td data-bbox="911 1665 1481 1740">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="786 1740 911 1814">bash</td> <td data-bbox="911 1740 1481 1814">cd /bootflash;mkdir new_dir</td> </tr> </table>	cli_show	show version ; show interface brief ; show vlan	cli_conf	interface Eth4/1 ; no shut ; switchport	bash	cd /bootflash;mkdir new_dir
cli_show	show version ; show interface brief ; show vlan						
cli_conf	interface Eth4/1 ; no shut ; switchport						
bash	cd /bootflash;mkdir new_dir						

NX-API Request Element	Description				
<i>output_format</i>	The available output message formats are the following:				
	<table border="1"> <tr> <td data-bbox="824 344 1062 394">xml</td> <td data-bbox="1062 344 1515 394">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="824 394 1062 453">json</td> <td data-bbox="1062 394 1515 453">Specifies output in JSON format.</td> </tr> </table>	xml	Specifies output in XML format.	json	Specifies output in JSON format.
	xml	Specifies output in XML format.			
json	Specifies output in JSON format.				
<p>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.</p> <p>To manage the computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported.</p> <p>The content-type header in the HTTP/HTTPS headers indicate the type of response format (XML or JSON).</p>					

NX-API Response Elements

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

Table 3: NX-API Response Elements

NX-API Response Element	Description
version	NX-API version.
type	Type of command to be executed.
sid	Session ID of the response. This element is valid only when the response message is chunked.
outputs	<p>Tag that encloses all command outputs.</p> <p>When multiple commands are in <code>cli_show</code> or <code>cli_show_ascii</code>, each command output is enclosed by a single output tag.</p> <p>When the message type is <code>cli_conf</code> or <code>bash</code>, there is a single output tag for all the commands because <code>cli_conf</code> and <code>bash</code> commands require context.</p>
output	<p>Tag that encloses the output of a single command output.</p> <p>For <code>cli_conf</code> and <code>bash</code> message types, this element contains the outputs of all the commands.</p>
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-Dispute"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

XML and JSON Supported Commands

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

- XML

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

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

Selected examples of this feature follow.

Examples of XML and JSON Output

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

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096",
 "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "used_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_total": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "used_mcast_oif1": "2", "used_host_in_host_total": "13", "used_host4_in_host": "12", "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://www.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>
                </_readonly__>
              </_XML_OPT_Cmd_dynamic_tcam_status__readonly__>
            </_XML_OPT_Cmd_dynamic_tcam_status>
          </status>
        </profile>
      </hardware>
    </show>
  </nf:data>
</nf:rpc-reply>
```

```

<used_v6_lpm_128>1</used_v6_lpm_128>
<used_host_lpm_total>0</used_host_lpm_total>
<used_host_v4_lpm>0</used_host_v4_lpm>
<used_host_v6_lpm>0</used_host_v6_lpm>
<used_mcast>0</used_mcast>
<used_mcast_oif1>2</used_mcast_oif1>
<used_host_in_host_total>13</used_host_in_host_total>
<used_host4_in_host>12</used_host4_in_host>
<used_host6_in_host>1</used_host6_in_host>
<max_ecmp_table_limit>64</max_ecmp_table_limit>
<used_ecmp_table>0</used_ecmp_table>
<mfib_fd_status>Disabled</mfib_fd_status>
<mfib_fd_maxroute>0</mfib_fd_maxroute>
<mfib_fd_count>0</mfib_fd_count>
</__readonly__>
</__XML_OPT_Cmd_dynamic_tcam_status__readonly__>
</__XML_OPT_Cmd_dynamic_tcam_status>
</status>
</profile>
</hardware>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#

```

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

```

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

```

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

```

switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:lldp">
  <nf:data>
    <show>
      <lldp>
        <timers>
          <__XML_OPT_Cmd_lldp_show_timers__readonly__>
            <__readonly__>
              <ttl>120</ttl>
              <reinit>2</reinit>
              <tx_interval>30</tx_interval>
              <tx_delay>2</tx_delay>
              <hold_mplier>4</hold_mplier>
              <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:params:xml:ns:netconf:base:1.0">
  <nf:data>
    <show>
      <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
        <ip_ipv6_mac>ip</ip_ipv6_mac>
        <access-lists>
          <__XML__OPT_Cmd_show_acl_name>
            <name>acl-test1</name>
          <__XML__OPT_Cmd_show_acl_capture>
            <__XML__OPT_Cmd_show_acl_expanded>
              <__XML__OPT_Cmd_show_acl__readonly__>
                <__readonly__>
                  <TABLE_ip_ipv6_mac>
                    <ROW_ip_ipv6_mac>
                      <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
                      <show_summary>0</show_summary>
                      <acl_name>acl-test1</acl_name>
                      <statistics>enable</statistics>
                      <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
                    <TABLE_seqno>
                      <ROW_seqno>
                        <seqno>10</seqno>
                        <permitdeny>permit</permitdeny>
                        <ip>ip</ip>
                        <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
                        <dest_any>any</dest_any>
                      </ROW_seqno>
                    </TABLE_seqno>
                  </ROW_ip_ipv6_mac>
                </TABLE_ip_ipv6_mac>
              </__readonly__>
            </__XML__OPT_Cmd_show_acl__readonly__>
          </__XML__OPT_Cmd_show_acl_expanded>
        </__XML__OPT_Cmd_show_acl_capture>
      </__XML__OPT_Cmd_show_acl_name>
    </access-lists>
  </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#
```

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

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

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

```
switch-1# show system redundancy status | json-pretty native
{
  "rdn_mode_admin":      "HA",
  "rdn_mode_oper":      "None",
```

```

    "this_sup":      "(sup-1)",
    "this_sup_rdn_state": "Active, SC not present",
    "this_sup_sup_state": "Active",
    "this_sup_internal_state": "Active with no standby",
    "other_sup":     "(sup-1)",
    "other_sup_rdn_state": "Not present"
  }
switch-1#

```

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

```

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

```

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

```

switch-1# show ip route summary | xml
<?xml version="1.0" encoding="ISO-8859-1"?> <nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:urib" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nf:data>
    <show>
      <ip>
        <route>
          <_XML_OPT_Cmd_urib_show_ip_route_command_ip>
            <_XML_OPT_Cmd_urib_show_ip_route_command_unicast>
              <_XML_OPT_Cmd_urib_show_ip_route_command_topology>
                <_XML_OPT_Cmd_urib_show_ip_route_command_l3vm-info>
                  <_XML_OPT_Cmd_urib_show_ip_route_command_rpf>
                    <_XML_OPT_Cmd_urib_show_ip_route_command_ip-addr>
                      <_XML_OPT_Cmd_urib_show_ip_route_command_protocol>
                        <_XML_OPT_Cmd_urib_show_ip_route_command_summary>
                          <_XML_OPT_Cmd_urib_show_ip_route_command_vrf>
                            <_XML_OPT_Cmd_urib_show_ip_route_command__readonly__>
                              <_readonly__>
                                <TABLE_vrf>
                                  <ROW_vrf>
                                    <vrf-name-out>default</vrf-name-out>
                                    <TABLE_addrf>
                                      <ROW_addrf>
                                        <addrf>ipv4</addrf>
                                        <TABLE_summary>
                                          <ROW_summary>
                                            <routes>938</routes>
                                            <paths>1453</paths>
                                            <TABLE_unicast>
                                              <ROW_unicast>
                                                <clientnameuni>am</clientnameuni>
                                                <best-paths>2</best-paths>
                                              </ROW_unicast>
                                              <ROW_unicast>
                                                <clientnameuni>local</clientnameuni>
                                                <best-paths>105</best-paths>
                                              </ROW_unicast>
                                              <ROW_unicast>
                                                <clientnameuni>direct</clientnameuni>
                                                <best-paths>105</best-paths>
                                              </ROW_unicast>
                                              <ROW_unicast>
                                                <clientnameuni>broadcast</clientnameuni>

```

```

        <best-paths>203</best-paths>
    </ROW_unicast>
    <ROW_unicast>
        <clientnameuni>ospf-10</clientnameuni>
        <best-paths>1038</best-paths>
    </ROW_unicast>
</TABLE_unicast>
<TABLE_route_count>
<ROW_route_count>
    <mask_len>8</mask_len>
    <count>1</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>24</mask_len>
    <count>600</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>31</mask_len>
    <count>13</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>32</mask_len>
    <count>324</count>
</ROW_route_count>
</TABLE_route_count>
</ROW_summary>
</TABLE_summary>
</ROW_addrf>
</TABLE_addrf>
</ROW_vrf>
</TABLE_vrf>
</_readonly_>
</_XML_OPT_Cmd_urib_show_ip_route_command_readonly_>
</_XML_OPT_Cmd_urib_show_ip_route_command_vrf>
</_XML_OPT_Cmd_urib_show_ip_route_command_summary>
</_XML_OPT_Cmd_urib_show_ip_route_command_protocol>
</_XML_OPT_Cmd_urib_show_ip_route_command_ip-addr>
</_XML_OPT_Cmd_urib_show_ip_route_command_rpf>
</_XML_OPT_Cmd_urib_show_ip_route_command_l3vm-info>
</_XML_OPT_Cmd_urib_show_ip_route_command_topology>
</_XML_OPT_Cmd_urib_show_ip_route_command_unicast>
</_XML_OPT_Cmd_urib_show_ip_route_command_ip>
</route>
</ip>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1#

```

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

```

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

```

```

"true", "gr_planned_only": "true", "gr_grace_period": "PT60S", "gr_state": "inactive"
, "gr_last_status": "None", "support_tos0_only": "true", "support_opaque_lsa": "true"
, "is_abr": "false", "is_asbr": "false", "admin_dist": 110, "ref_bw": 40000, "spf_start
time": "PT0S", "spf_hold_time": "PT1S", "spf_max_time": "PT5S", "lsa_start_time": "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": "l11", "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": "l12", "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#

```