



NX-API CLI

This chapter contains the following topics:

- [About NX-API CLI, on page 1](#)
- [Using NX-API CLI, on page 2](#)
- [JSON and XML Structured Output, on page 16](#)

About NX-API CLI

NX-API CLI is an enhancement to the Cisco Nexus 3400-S platform switch CLI system, which supports XML output. NX-API CLI also supports JSON output format for specific commands.

On switches, command-line interfaces (CLIs) are run only on the device. NX-API CLI improves the accessibility of these CLIs by making them available outside of the switch by using HTTP or 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.

Transport

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

The NX-API feature is enabled by default on HTTPS port 443. HTTP port 80 is disabled.

NX-API is also supported through UNIX Domain Sockets for applications running natively on the host or within Guest Shell.

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

Message Format

NX-API is an enhancement to the Cisco NX-OS CLI system, which supports XML output. NX-API also supports JSON output format for specific commands.

**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 does not support insecure HTTP by default.
- NX-API does not support weak TLSv1 protocol by default.

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. NX-API encodes the CLIs into the body of an HTTP or HTTPS POST request. 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 15](#).

NX-API CLI is enabled by default for local access. The remote HTTP access is disabled by default.

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 management
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"
      }
    }
  }
}

```



Note There is a known issue where an attempt to delete a user might fail. The result is an error similar to the following appearing approximately every 12 hours:

```
user delete failed for username:userdel: user username is currently logged in - securityd
```

This issue might occur in a scenario where you try to delete a user who is still logged into a switch through NX-API. Enter the following command in this case to try to log the user out first:

```
switch(config)# clear user username
```

Then try to delete the user again. If the issue persists after attempting this workaround, contact Cisco TAC for further assistance.

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. 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>

```

Sample NX-API Scripts

The sample scripts demonstrate how a script is used with NX-API.

- Cable Checker (check_cable.py)
- Cable Checker Blueprint (connectivity.json)
- Using NX-API over UDS (rest_client.py)

NX-API Management Commands

The following table shows the CLI commands that can manage and enable through the NX-API.

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 <i>port</i>	Specifies a port.
no nxapi {http https}	Disables HTTP or HTTPS.
show nxapi	Displays port information.
nxapi certificate {httpsrct certfile httpskey keyfile} <i>filename</i>	<p>Specifies the upload of the following:</p> <ul style="list-style-type: none"> • HTTPS certificate when <code>httpsrct</code> is specified. • HTTPS key when <code>httpskey</code> is specified. <p>Example of an HTTPS certificate:</p> <pre>nxapi certificate httpsrct certfile bootflash:cert.crt</pre> <p>Example of an HTTPS key:</p> <pre>nxapi certificate httpskey keyfile bootflash:privkey.key</pre>
nxapi certificate enable	Enables a certificate.
nxapi ssl-ciphers weak	Weak ciphers are disabled by default. Running this command changes the default behavior and enables the weak ciphers for NGINX. The no form of the command changes it to the default (by default, the weak ciphers are disabled).
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	TLS1.0 is disabled by default. Running this command enables the TLS versions that are specified in the string, including the TLS1.0 that was disabled by default, if necessary. The no form of the command changes it to the default (by default, only TLS1.1 and TLS1.2 are enabled).
nxapi use-vrf <i>vrf</i>	Specifies the default VRF, management VRF, or named VRF.
ip netns exec management iptables	<p>Implements any access restrictions and can be run in management VRF.</p> <p>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.</p> <p><i>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.</i></p>

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

```
switch(config)# nxapi certificate https crt 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 can 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 error can occur if the key file is encrypted. In that case, you must decrypt the key file 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 the Guest Shell chapter in this document 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 time outs 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 that use **terminal dont-ask** 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 sends request elements 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.
<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_array CLI show commands that expect structured output. Only for show commands. Similar to cli_show, but with cli_show_array, data is returned as a list of one element, or an array, within square brackets []. • cli_show_ascii CLI show commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes. • cli_conf CLI configuration commands. • bash Bash commands. Most non-interactive Bash commands are supported by NX-API. <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="930 485 1516 596"> <tr> <td data-bbox="930 485 1024 539">Note</td> <td data-bbox="1024 485 1105 539">0</td> <td data-bbox="1105 485 1516 539">Do not chunk output.</td> </tr> <tr> <td data-bbox="930 539 1024 596"></td> <td data-bbox="1024 539 1105 596">1</td> <td data-bbox="1105 539 1516 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>	Note	0	Do not chunk output.		1	Chunk output.
Note	0	Do not chunk output.					
	1	Chunk output.					
<i>rollback</i>	<p>Valid only for configuration CLIs, not for show commands. Specifies the configuration rollback options. Specify one of the following options.</p> <ul style="list-style-type: none"> • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in. <p>Note The rollback element is available in the cli_conf mode when the input request format is XML or JSON.</p>						
<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>						

NX-API Request Element	Description						
<p><i>input</i></p>	<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>Prepend commands with terminal dont-ask to avoid timing out with an error code 500. For example:</p> <pre>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</pre> <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="786 863 1479 1087"> <tr> <td data-bbox="786 863 911 936">cli_show</td> <td data-bbox="911 863 1479 936">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="786 936 911 1010">cli_conf</td> <td data-bbox="911 936 1479 1010">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="786 1010 911 1087">bash</td> <td data-bbox="911 1010 1479 1087">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						
<p><i>output_format</i></p>	<p>The available output message formats are the following:</p> <p>Note</p> <table border="1" data-bbox="891 1163 1479 1276"> <tr> <td data-bbox="891 1163 1092 1220">xml</td> <td data-bbox="1092 1163 1479 1220">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="891 1220 1092 1276">json</td> <td data-bbox="1092 1220 1479 1276">Specifies output in JSON format.</td> </tr> </table> <p>Note The Cisco NX-OS 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>	xml	Specifies output in XML format.	json	Specifies output in JSON format.		
xml	Specifies output in XML format.						
json	Specifies output in JSON format.						

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

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

NX-API Request Element	Description
<i>jsonrpc</i>	A string specifying the version of the JSON-RPC protocol. Version must be 2.0.
<i>method</i>	A string containing the name of the method to be invoked. NX-API supports either: <ul style="list-style-type: none"> • cli—show or configuration commands • cli_ascii—show or configuration commands; output without formatting • cli_array—only for show commands; similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [].
<i>params</i>	A structured value that holds the parameter values used during the invocation of a method. It must contain the following: <ul style="list-style-type: none"> • cmd—CLI command • version—NX-API request version identifier
<i>rollback</i>	Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options. <ul style="list-style-type: none"> • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores the failed CLI and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
<i>id</i>	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, <i>id</i> : 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. 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. 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.
output	Tag that encloses the output of a single command output. For <code>cli_conf</code> and <code>bash</code> message types, this element contains the outputs of all the commands.
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution. NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml).
msg	Error message that is 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 a VRF for NX-API communication. For information about configuring ACLs, see the *Cisco Nexus 3400-S NX-OS Security Configuration Guide*.

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 13](#).

Updating an iptable

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

Procedure

Step 1 To create a rule that blocks HTTP access:

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

Step 2 To create a rule that blocks HTTPS access:

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

Step 3 To verify the applied rules:

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

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

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           tcp dpt:http
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 you modify them in the Bash shell. To make the rules persistent, see [Making an Iptable Persistent Across Reloads, on page 14](#).

Making an Iptable Persistent Across Reloads

The rules in iptables are not persistent across reloads when you modify them in the Bash shell. This section explains how to make a modified iptable persistent across a reload.

Before you begin

You have modified an iptable.

Procedure

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

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

Step 2 Create 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 3 Set the appropriate permissions to the startup script:

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

Step 4 Set the iptables_init startup script to on with the chkconfig utility:

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

The iptables_init startup script now executes 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.
IN_MSG_ERR	400	Request message is invalid.

NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.
CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow show .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive show commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Backend processing error.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

JSON and XML Structured Output

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

- XML
- JSON
- JSON Pretty, which makes the standard block of JSON-formatted output easier to read.

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

Selected examples of this feature follow.

About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard that is designed for human-readable data and is an alternative to XML. JSON originally evolved from JavaScript, but it is a language-independent data format. Command output from NX-OS supports JSON and JSON Pretty format.

The switch supports JSON CLI Execution.

Nearly all modern programming languages support two primary Data Structures in some way. These Data Structures are:

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

Also, through the Cisco NX-OS Sandbox, you can access JSON or XML output for a **show** command.

CLI Execution

```
switch-1# 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"}]}}
switch-1#
```

Examples of XML and JSON Output

This section documents selected examples of NX-OS commands that are displayed as 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>
                  <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 that are configured on the switch in JSON format:

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

This example shows how to display LLDP timers that are 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__>
              <t1>120</t1>
              <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)#
```

