



## NX-API Developer Sandbox

- [NX-API Developer Sandbox: NX-OS Releases Prior to 9.2\(2\), on page 1](#)
- [NX-API Developer Sandbox: NX-OS Release 9.2\(2\) and Later, on page 13](#)

## NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2)

### About the NX-API Developer Sandbox

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

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

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

Controls in the Command pane allow you to choose a message format for a supported API, such as NX-API REST, and a command type, such as XML or JSON. The available command type options vary depending on the selected message format.

When you type or paste one or more CLI commands into the Command pane, the web form converts the commands into an API payload, checking for configuration errors, and displays the resulting payload in the Request pane. If you then choose to post the payload directly from the Sandbox to the switch, using the POST button in the Command pane, the Response pane displays the API response.

## Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled. For example, configuring a BGP router requires first enabling BGP with the **feature bgp** command. Similarly, configuring an OSPF router requires first enabling OSPF with the **feature ospf** command. This also applies to **evpn esi multihoming**, which enables its dependent commands such as **evpn multihoming core-tracking**. For more information about enabling features to access feature dependent commands, see the .
- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- When you use XML or JSON as the Message Type for CLI input, you can use semicolon to separate multiple commands on the same line. However, when you use JSON RPC as the Message Type for CLI input, you cannot enter multiple commands on the same line and separate them with a semicolon (;).

For example, assume that you want to send **show hostname** and **show clock** commands through JSON RPC as the following.

In the Sandbox, you enter the CLIs as follows.

```
show hostname ; show clock
```

In the JSON RPC request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname ; show clock",
      "version": 1
    },
    "id": 1
  }
]
```

When you send the request, the response returns the following error.

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
      "msg": "Request contains invalid special characters"
    }
  },
}
```

```
    "id": 1
  }
}
```

This situation occurs because the Sandbox parses each command in a JSON RPC request as individual items and assigns an ID to each. When using JSON RPC requests, you cannot use internal punctuation to separate multiple commands on the same line. Instead, enter each command on a separate line and the request completes successfully.

Continuing with the same example, enter the commands as follows in the NX-API CLI.

```
show hostname
show clock
```

In the request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]
```

The response completes successfully.

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

## Configuring the Message Format and Command Type

The **Message Format** and **Command Type** are configured in the upper right corner of the Command pane (the top pane). For **Message Format**, choose the format of the API protocol that you want to use. The Developer Sandbox supports the following API protocols:

**Table 1: NX-OS API Protocols**

| Protocol    | Description  |
|-------------|--|
| json-rpc    | A standard lightweight remote procedure call (RPC) protocol that can be used to deliver NX-OS CLI commands in a JSON payload. The JSON-RPC 2.0 specification is outlined by <a href="https://jsonrpc.org">jsonrpc.org</a> .  |
| xml         | Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML payload.   |
| json        | Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in a JSON payload.   |
| nx-api rest | Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see <a href="https://developer.cisco.com/site/cisco-nexus-nx-api-references/">https://developer.cisco.com/site/cisco-nexus-nx-api-references/</a> . |
| nx yang     | The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.  |

When the **Message Format** has been chosen, a set of **Command Type** options are presented just below the **Message Format** control. The **Command Type** setting can constrain the input CLI and can determine the **Request** and **Response** format. The options vary depending on the **Message Format** selection. For each **Message Format**, the following table describes the **Command Type** options:

**Table 2: Command Types**

| Message format | Command type   |
|----------------|--|
| json-rpc       | <ul style="list-style-type: none"><li>cli — show or configuration commands</li><li>cli-ascii — show or configuration commands, output without formatting</li></ul>   |
| xml            | <ul style="list-style-type: none"><li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li><li>cli_show_ascii — show commands, output without formatting</li><li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li><li>bash — bash commands. Most non-interactive bash commands are supported.</li></ul> <p><b>Note</b><br/>The bash shell must be enabled in the switch.</p> |

| Message format | Command type  |
|----------------|---|
| json           | <ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b><br/>The bash shell must be enabled in the switch.</p> |
| nx-api rest    | <ul style="list-style-type: none"> <li>cli — configuration commands</li> </ul>  |
| nx yang        | <ul style="list-style-type: none"> <li>json — JSON structure is used for payload</li> <li>xml — XML structure is used for payload</li> </ul>  |

### Output Chunking

In order to handle large show command output, some NX-API message formats support output chunking for show commands. In this case, an **Enable chunk mode** checkbox appears below the **Command Type** control along with a session ID (**SID**) type-in box.

When chunking is enabled, the response is sent in multiple "chunks," with the first chunk sent in the immediate command response. In order to retrieve the next chunk of the response message, you must send an NX-API request with **SID** set to the session ID of the previous response message.

## Using the Developer Sandbox

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


**Tip**

Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window. Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI". Only configuration commands are supported.

### Procedure

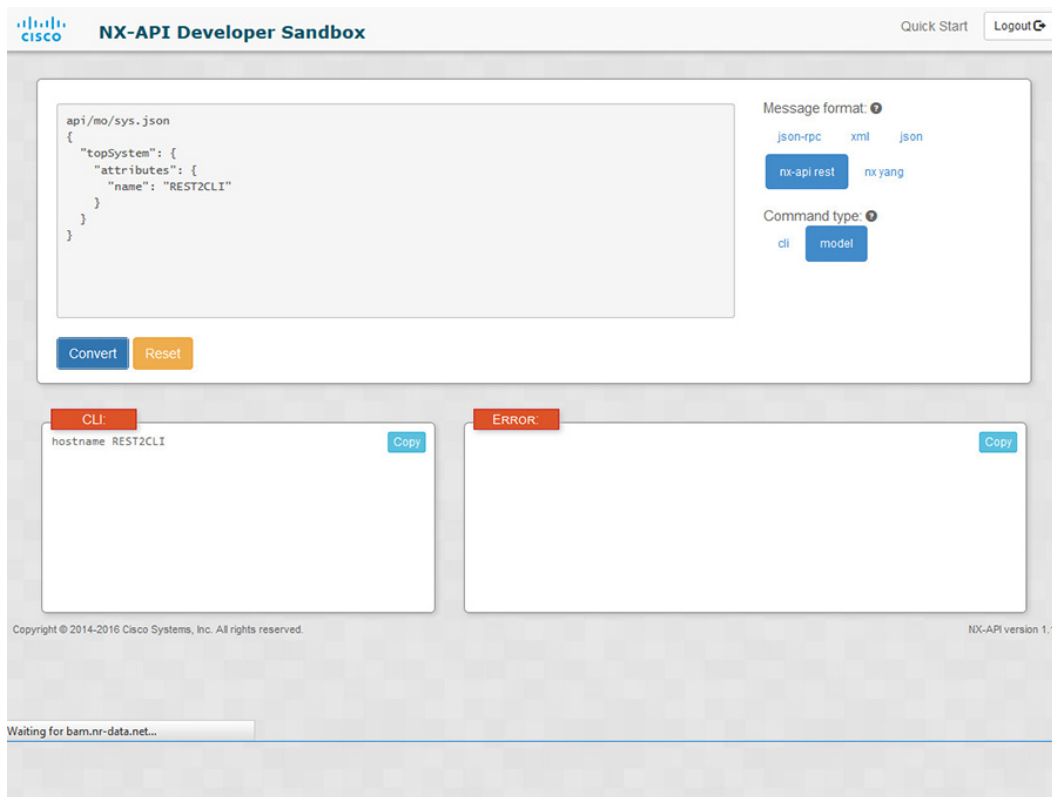
- Step 1** Configure the **Message Format** and **Command Type** for the API protocol you want to use.  
For detailed instructions, see [Configuring the Message Format and Command Type, on page 4](#).
- Step 2** Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.

You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.

The screenshot shows the NX-API Developer Sandbox interface. At the top, there is a header with the Cisco logo, the title "NX-API Developer Sandbox", and links for "Quick Start" and "Logout". The main area is divided into two sections. The top section contains a large text entry box with the placeholder text "Enter CLI commands here, one command per line." To the right of this box are two dropdown menus: "Message format:" with options "json-rpc", "xml", "json", "nx-api rest" (selected), and "nx yang"; and "Command type:" with options "cli" and "model" (selected). Below the text entry box are two buttons: "Convert" (blue) and "Reset" (orange). The bottom section is divided into two panes. The left pane is labeled "CLI" and contains a large empty text area with a "Copy" button. The right pane is labeled "ERROR" and also contains a large empty text area with a "Copy" button. At the bottom of the interface, there is a copyright notice: "Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved." and the version number "NX-API version 1.1".

**Step 3** Click the **Convert** at the bottom of the top pane.

If the CLI commands contain no configuration errors, the payload appears in the **Request** pane. If errors are present, a descriptive error message appears in the **Response** pane.

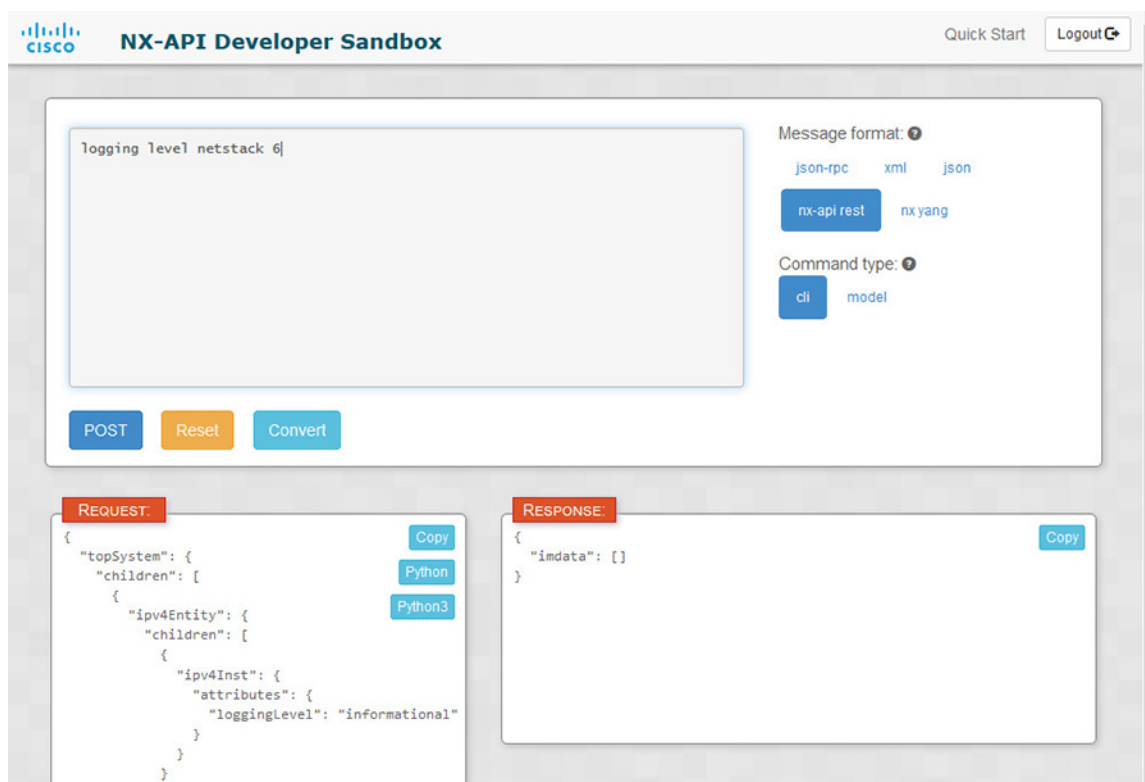


**Step 4** When a valid payload is present in the **Request** pane, you can click **POST** to send the payload as an API call to the switch.

The response from the switch appears in the **Response** pane.

**Warning**

Clicking **POST** commits the command to the switch, which can result in a configuration or state change.



**Step 5** You can copy the contents of the **Request** or **Response** pane to the clipboard by clicking **Copy** in the pane.

**Step 6** You can obtain a Python implementation of the request on the clipboard by clicking **Python** in the **Request** pane.

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



**Tip** Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window. Additional details, such as response codes and security methods, can be found in the chapter "NX-API CLI".

### SUMMARY STEPS

1. Select **nx-api rest** as the **Message Format** and **model** as the **Command Type**.
2. Enter a DN and payload into the text entry box in the top pane. Then click on the **Convert** button below the top pane.

### DETAILED STEPS

#### Procedure

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



**Example:**

Enter CLI commands here, one command per line.

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

Command type:   
cli model

Convert Reset

CLI Copy

ERROR Copy

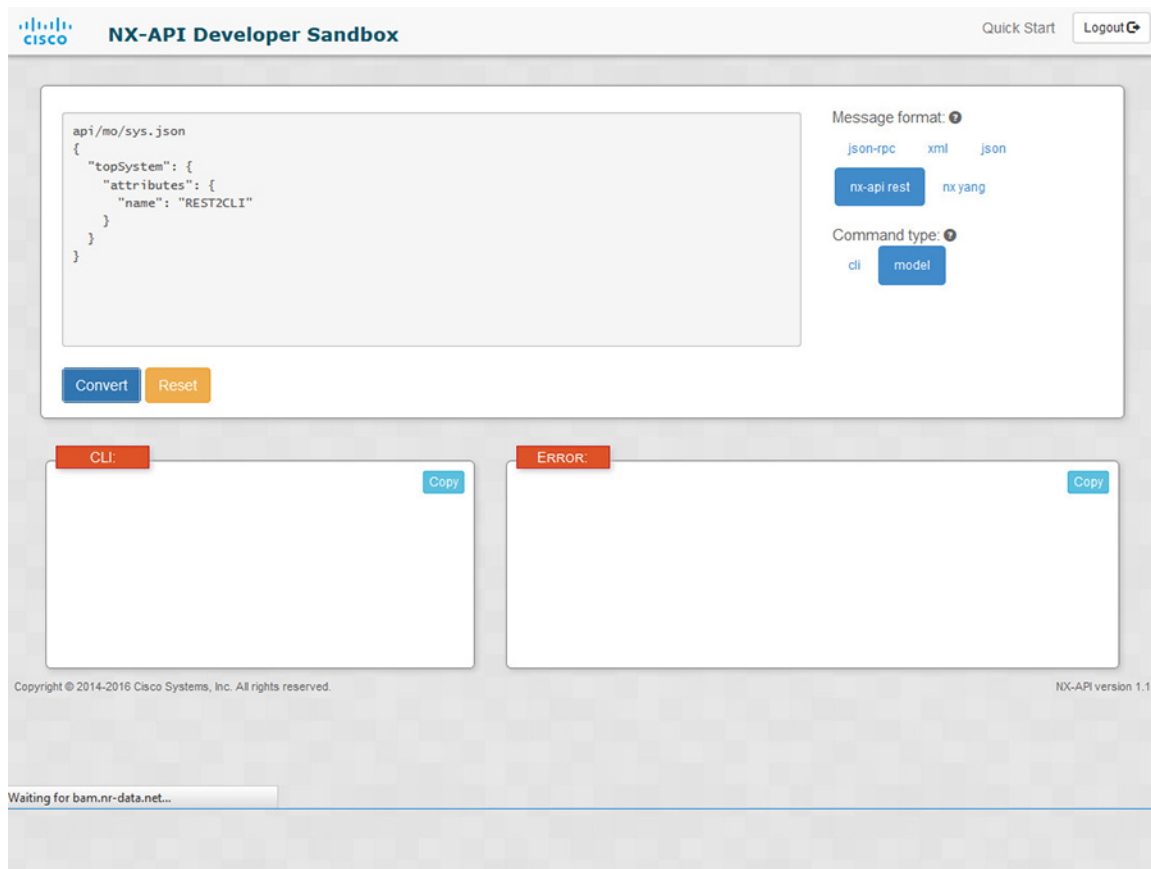
Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved. NX-API version 1.1

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


**Example:**

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

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





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

 **NX-API Developer Sandbox**

Quick Start [Logout](#)

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

Message format:   
[json-rpc](#) [xml](#) [json](#)  
[nx-api rest](#) [nx yang](#)

Command type:   
[cli](#) [model](#)

[Convert](#) [Reset](#)

CLI:

hostname REST2CLI [Copy](#)

ERROR:

[Copy](#)

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

NX-API version 1.1

Waiting for bam.nr-data.net...

**Note**

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

Table 3: Sources of REST2CLI Errors

| Payload Issue   | Result  |
|---|---|
| <p>The payload contains an attribute that does not exist in the MO.</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "interfaceEntity": {           "children": [             {               "l1PhysIf": {                 "attributes": {                   "id": "eth1/1",                   "fakeattribute": "totallyFake"                 }               }             }           ]         }       }     ]   } }</pre> | <p>The <b>Error</b> pane will return an error related to the attribute.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> unknown attribute 'fakeattribute' in element 'l1PhysIf'</p> |
| <p>The payload includes MOs that aren't yet supported for conversion:</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "dhcpEntity": {           "children": [             {               "dhcpInst": {                 "attributes": {                   "SnoopingEnabled": "yes"                 }               }             }           ]         }       }     ]   } }</pre>  | <p>The <b>Error</b> Pane will return an error related to the unsupported MO.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> The entire subtree of "sys/dhcp" is not converted.</p> |

# NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later

## About the NX-API Developer Sandbox

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

The web form is a single screen with three panes — Command (top pane), Request (middle pane), and Response (bottom pane) — as shown in the figure below. The designated name (DN) field is located between the Command and Request panes (seen in the figure below located between the **POST** and **Send** options).

The Request pane also has a series of tabs. Each tab represents a different language: **Python**, **Python3**, **Java**, **JavaScript**, and **Go-Lang**. Each tab enables you to view the request in the respective language. For example, after converting CLI commands into an XML or JSON payload, click the **Python** tab to view the request in Python, which you can use to create scripts.

**Figure 2: NX-API Developer Sandbox with Example Request and Output Response**

The screenshot displays the NX-API Developer Sandbox interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title, along with links for 'Quick Start', 'Command Reference', and 'Logout'. The main area is divided into three panes. The top pane, 'Command', contains the text 'show version'. To its right are dropdown menus for 'Method' (set to 'NX-API-CLI'), 'Message format' (set to 'json-rpc'), and 'Input type' (set to 'cli\_ascii'). Below the Command pane is a 'POST' button and a text field containing '/ins'. To the right of this field are 'Send', 'Reset', and 'Output Schema' buttons. The middle pane, 'Request', has tabs for 'Request', 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is active, showing a JSON payload: 

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

 A 'Copy' button is next to the payload. The bottom pane, 'Response', shows the resulting JSON response: 

```
{
  "jsonrpc": "2.0",
  "result": {
    "msg": "Cisco Nexus Operating System (NX-OS) Software\nTAC support: http://www.cisco.com/tac\nDocuments: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home."
  },
  "id": 1
}
```

 A 'Copy' button is also present next to the response.

Controls in the Command pane enable you to choose a supported API, such as NX-API REST, an input type, such as model (payload) or CLI, and a message format, such as XML or JSON. The available options vary depending on the chosen method.

When you choose the NX-API REST (DME) method, type or paste one or more CLI commands into the Command pane, and click **Convert**, the web form converts the commands into a REST API payload, checking for configuration errors, and displays the resulting payload in the Request pane. If you then choose to post

the payload directly from the sandbox to the switch (by choosing the **POST** option and clicking **SEND**), the Response pane displays the API response. For more information, see [Using the Developer Sandbox to Convert CLI Commands to REST Payloads, on page 19](#)

Conversely, the Cisco NX-API Developer Sandbox checks the payload for configuration errors then displays the equivalent CLIs in the Response pane. For more information, see [Using the Developer Sandbox to Convert from REST Payloads to CLI Commands, on page 22](#)

## Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled. For example, configuring a BGP router requires first enabling BGP with the **feature bgp** command. Similarly, configuring an OSPF router requires first enabling OSPF with the **feature ospf** command. This also applies to **evpn esi multihoming**, which enables its dependent commands such as **evpn multihoming core-tracking**. For more information about enabling features to access feature dependent commands, see the .
- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- When you use XML or JSON as the Message Type for CLI input, you can use semicolon to separate multiple commands on the same line. However, when you use JSON RPC as the Message Type for CLI input, you cannot enter multiple commands on the same line and separate them with a semicolon ( ; ).

For example, assume that you want to send **show hostname** and **show clock** commands through JSON RPC as the following.

In the Sandbox, you enter the CLIs as follows.

```
show hostname ; show clock
```

In the JSON RPC request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname ; show clock",
      "version": 1
    },
    "id": 1
  }
]
```

When you send the request, the response returns the following error.

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
```

```

      "msg": "Request contains invalid special characters"
    }
  },
  "id": 1
}

```

This situation occurs because the Sandbox parses each command in a JSON RPC request as individual items and assigns an ID to each. When using JSON RPC requests, you cannot use internal punctuation to separate multiple commands on the same line. Instead, enter each command on a separate line and the request completes successfully.

Continuing with the same example, enter the commands as follows in the NX-API CLI.

```

show hostname
show clock

```

In the request, the input is formatted as follows.

```

[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]

```

The response completes successfully.

```

[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]

```

## Configuring the Message Format and Input Type

The **Method**, **Message format**, and **Input type** are configured in the upper right corner of the Command pane (the top pane). For **Method**, choose the format of the API protocol that you want to use. The Cisco NX-API Developer Sandbox supports the following API protocols:

*Table 4: NX-OS API Protocols*

| Protocol         | Description  |
|------------------|--|
| NXAPI-CLI        | Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML or a JSON payload.   |
| NXAPI-REST (DME) | <p>Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. The NXAPI-REST (DME) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"><li>• <b>POST</b></li><li>• <b>GET</b></li><li>• <b>PUT</b></li><li>• <b>DELETE</b></li></ul> <p>For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see <a href="https://developer.cisco.com/site/cisco-nexus-nx-api-references/">https://developer.cisco.com/site/cisco-nexus-nx-api-references/</a>.</p> |
| RESTCONF (Yang)  | <p>The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.</p> <p>The RESTCONF (Yang) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"><li>• <b>POST</b></li><li>• <b>GET</b></li><li>• <b>PUT</b></li><li>• <b>PATCH</b></li><li>• <b>DELETE</b></li></ul>   |

When you choose the **Method**, a set of **Message format** or **Input type** options are displayed in a drop-down list. The **Message format** can constrain the input CLI and determine the **Request** and **Response** format. The options vary depending on the **Method** you choose.

The following table describes the **Input/Command type** options for each **Message format**:



Table 5: Command Types

| Method    | Message format | Input/Command type   |
|-----------|----------------|--|
| NXAPI-CLI | json-rpc       | <ul style="list-style-type: none"> <li>cli — show or configuration commands</li> <li>cli-ascii — show or configuration commands, output without formatting</li> <li>cli-array — show commands. Similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [ ].</li> </ul>   |
| NXAPI-CLI | xml            | <ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b><br/>The bash shell must be enabled in the switch.</p>  |
| NXAPI-CLI | json           | <ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> </ul> <p><b>Note</b><br/>Beginning with Cisco NX-OS Release 9.3(3), the cli_show_array command is recommended over the cli_show command.</p> <ul style="list-style-type: none"> <li>cli_show_array — 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 [ ].</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b><br/>The bash shell must be enabled in the switch.</p> |

| Method           | Message format   | Input/Command type  |
|------------------|--|---|
| NXAPI-REST (DME) |  | <ul style="list-style-type: none"> <li>cli — CLI to model conversion</li> <li>model — Model to CLI conversion.</li> </ul> |
| RESTCONF (Yang)  | <ul style="list-style-type: none"> <li>json — JSON structure is used for payload</li> <li>xml — XML structure is used for payload</li> </ul> |   |

### Output Chunking

JSON and XML NX-API message formats enable you to receive large show command responses in 10-MB chunks. When received, the chunks are concatenated to create a valid JSON object or XML structure. To view a sample script that demonstrates output chunking, click the following link and choose the directory that corresponds to Release 9.3x: [Cisco NX-OS NXAPI](#).



**Note** For chunk JSON mode, the browser or python script part does not provide the valid JSON output (there will be no closing tags). To use chunk mode and get valid JSON, use the script provided in the directory.

You receive the first chunk in the immediate command response, which also includes a **sid** field that contains a session Id. To retrieve the next chunk, you enter the session Id from the previous chunk in the **SID** text box. You repeat the process until reaching the last response, which is indicated by the **eoc** (end of content) value in the **sid** field.

Chunk mode is available when using the **NXAPI-CLI** method with the **JSON** or **XML** format type and the **cli\_show**, **cli\_show\_array**, or **cli\_show\_ascii** command type. For more information about configuring the chunk mode, see the *Chunk Mode Fields* table.



**Note** NX-API supports a maximum of 2 chunking sessions.

**Table 6: Chunk Mode Fields**

| Field Name               | Description   |
|--------------------------|---|
| <b>Enable Chunk Mode</b> | Click to place a check mark in the <b>Enable Chunk Mode</b> check box to enable chunking. When you enable chunk mode, responses that exceed 10 MB are sent in multiple chunks of up to 10 MB in size. |

| Field Name | Description   |
|------------|---|
| <b>SID</b> | Enter the session Id of the previous response in the <b>SID</b> text box to retrieve the next chunk of the response message.<br><br><b>Note</b><br>Only alphanumeric characters and ‘_’ are allowed. Invalid characters receive an error. |

## Using the Developer Sandbox

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

**Tip**

- Online help is available by clicking the help icons (?) next to the field names located in the upper-right corner of the Cisco NX-API Developer Sandbox window.
- For additional details, such as response codes and security methods, see the *NX-API CLI* chapter.
- Only configuration commands are supported.

The Cisco NX-API Developer Sandbox enables you to convert CLI commands to REST payloads.

#### Procedure

- 
- Step 1** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.  
The **Input** type drop-down list appears.
- Step 2** Click the **Input** type drop-down list and choose **cli**.
- Step 3** Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.  
You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.

The screenshot shows the NX-API Sandbox web interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title. Navigation links include 'Quick Start', 'DME Documentation', 'Model Browser', and 'Logout'. A large text area on the left is labeled 'Enter DME payload here.' Below it, a text input field contains '/api/mo/sys.json'. To the right of this input are three buttons: 'Send' (blue), 'Reset' (orange), and 'Convert' (blue). Further right, there are two dropdown menus: 'Method:' set to 'NXAPI-REST (DME)' and 'Input type:' set to 'model'. Below the input field, there are tabs for 'Request', 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is active, showing a large empty text area with a 'Copy' button in the top right corner. Below the 'Request' tab is a 'Response:' section, also with a large empty text area and a 'Copy' button in the top right corner.

**Step 4** Click **Convert**.

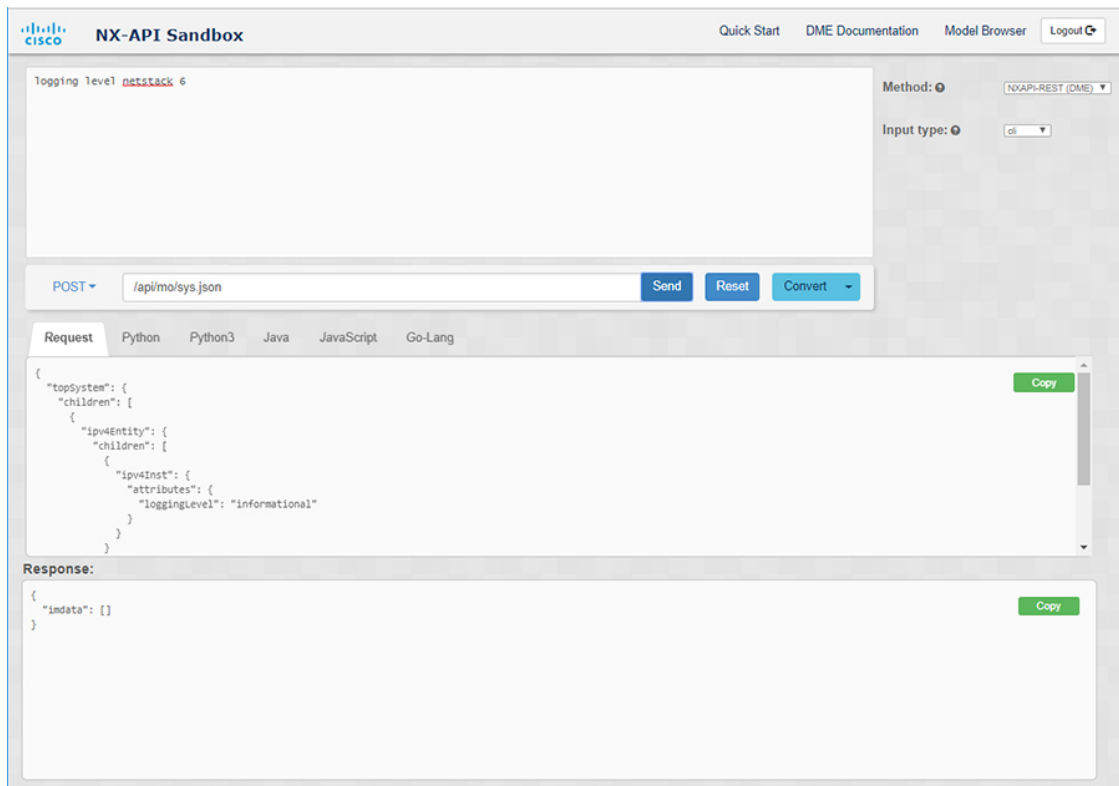
If the CLI commands contain no configuration errors, the payload appears in the **Request** pane. If errors are present, a descriptive error message appears in the **Response** pane.

**Step 5** (Optional) To send a valid payload as an API call to the switch, click **Send**.

The response from the switch appears in the **Response** pane.

**Warning**

Clicking **Send** commits the command to the switch, which can result in a configuration or state change.



**Step 6** (Optional) To obtain the DN for an MO in the payload:

- a. From the **Request** pane, choose **POST**.
- b. Click the **Convert** drop-down list and choose **Convert (with DN)**.

The payload appears with with a **dn** field that contains the DN that corresponds to each MO in the payload.

**Step 7** (Optional) To overwrite the current configuration with a new configuration:

- a. Click the **Convert** drop-down list and choose **Convert (for Replace)**. The **Request** pane displays a payload with a **status** field set to **replace**.
- b. From the **Request** pane, choose **POST**.
- c. Click **Send**.

The current configuration is replaced with the posted configuration. For example, if you start with the following configuration:

```
interface eth1/2
  description test
  mtu 1501
```

Then use **Convert (for Replace)** to POST the following configuration:

```
interface eth1/2
  description testForcr
```

The `mtu` configuration is removed and only the new description (`testForcr`) is present under the interface. This change is confirmed when entering **show running-config**.

**Step 8** (Optional) To copy the contents of a pane, such as the **Request** or **Response** pane, click **Copy**. The contents of the respective pane is copied to the clipboard.

**Step 9** (Optional) To convert the request into one of the formats listed below, click on the appropriate tab in the **Request** pane:

- **Python**
- **Python3**
- **Java**
- **JavaScript**
- **Go-Lang**

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

The Cisco NX-API Developer Sandbox enables you to convert REST payloads to corresponding CLI commands. This option is only available for the NXAPI-REST (DME) method.



### Tip

- Online help is available by clicking help icons (?) next to the Cisco NX-API Developer Sandbox field names. Click a help icon to get information about the respective field.

For additional details, such as response codes and security methods, see the chapter *NX-API CLI*.

- The top-right corner of the Cisco NX-API Developer Sandbox contains links for additional information. The links that appear depend on the **Method** you choose. The links that appear for the NXAPI-REST (DME) method:

- **NX-API References**—Enables you to access additional NX-API documentation.
- **DME Documentation**—Enables you to access the NX-API DME Model Reference page.
- **Model Browser**—Enables you to access Visore, the Model Browser. Note that you might have to manually enter the IP address for your switch to access the Visore page:

`https://management-ip-address/visore.html.`

## Procedure

**Step 1** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.

**Example:**

The screenshot shows the NX-API Sandbox web interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title, along with links for 'Quick Start', 'DME Documentation', 'Model Browser', and a 'Logout' button. Below the header, there's a large text area for 'Enter DME payload here.'. To the right of this area, there are two dropdown menus: 'Method:' set to 'NX-API-REST (DME)' and 'Input type:' set to 'model'. Below the payload area, there's a text input field containing '/api/mo/sys.json', and three buttons: 'Send' (blue), 'Reset' (orange), and 'Convert' (blue). Below these buttons, there are tabs for 'Request', 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is active, showing a large empty text area with a 'Copy' button in the top right corner. Below the 'Request' pane, there's a 'Response:' section with another large empty text area and a 'Copy' button in the top right corner.

**Step 2** Click the **Input Type** drop-down list and choose **model**.

**Step 3** Enter the designated name (DN) that corresponds to the payload in the field above the Request pane.

**Step 4** Enter the payload in the Command pane.

**Step 5** Click **Convert**.

**Example:**

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

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

The screenshot shows the NX-API Sandbox interface. At the top, there's a header with the Cisco logo and the title "NX-API Sandbox". Navigation links include "Quick Start", "DME Documentation", "Model Browser", and a "Logout" button. The main area is divided into two sections. The top section contains a JSON payload in a text area: 

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

 To the right of the text area are two dropdown menus: "Method:" set to "NX-API-REST (DME)" and "Input type:" set to "model". Below the text area is a URL input field containing "/api/mo/sys.json" and three buttons: "Send" (blue), "Reset" (orange), and "Convert" (blue). Below the URL field is a tabbed interface with tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing a large empty text area for the request body, with a "Copy" button in the top right corner. Below the request area is a "Response:" section, also with a large empty text area and a "Copy" button in the top right corner.

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



The screenshot displays the Cisco NX-API Developer Sandbox interface. At the top, there is a header with the Cisco logo, the title "NX-API Sandbox", and navigation links for "Quick Start", "DME Documentation", "Model Browser", and a "Logout" button. The main workspace is divided into two sections. The top section contains a JSON payload in a text area: 

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

. To the right of this area, there are controls for "Method" (set to "NX-API REST (DME)") and "Input type" (set to "model"). Below the JSON area is a text input field containing the path "/api/mo/sys.json", followed by "Send", "Reset", and "Convert" buttons. The bottom section is titled "Request" and shows the converted CLI command: "hostname REST2CLI". To the right of this command is a green "Copy" button. Below the request section is a "Response:" section, which is currently empty, with another green "Copy" button to its right. At the bottom of the "Request" section, there are tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang".

**Note**

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

Table 7: Sources of REST2CLI Errors

| Payload Issue   | Result  |
|---|---|
| <p>The payload contains an attribute that does not exist in the MO.</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "interfaceEntity": {           "children": [             {               "l1PhysIf": {                 "attributes": {                   "id": "eth1/1",                   "fakeattribute": "totallyFake"                 }               }             }           ]         }       }     ]   } }</pre> | <p>The <b>Error</b> pane will return an error related to the attribute.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> unknown attribute 'fakeattribute' in element 'l1PhysIf'</p> |
| <p>The payload includes MOs that aren't yet supported for conversion:</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "dhcpEntity": {           "children": [             {               "dhcpInst": {                 "attributes": {                   "SnoopingEnabled": "yes"                 }               }             }           ]         }       }     ]   } }</pre>  | <p>The <b>Error</b> Pane will return an error related to the unsupported MO.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> The entire subtree of "sys/dhcp" is not converted.</p> |

## Using the Developer Sandbox to Convert from RESTCONF to json or XML

**Tip**

- Online help is available by clicking the help icon (?) in the upper-right corner of the Cisco NX-API Developer Sandbox window.
- Click on the **Yang Documentation** link in the upper right corner of the Sandbox window to go to the Model Driven Programmability with Yang page.
- Click on the **Yang Models** link in the upper right corner of the Sandbox window to access the YangModels GitHub site.

### Procedure

**Step 1** Click the **Method** drop-down list and choose **RESTCONF (Yang)**.

**Example:**

The screenshot shows the Cisco NX-API Sandbox interface. At the top, there's a header with the Cisco logo, 'NX-API Sandbox', and navigation links: 'Quick Start', 'Yang Documentation', 'Yang Models', and 'Logout'. Below the header, there's a text input area containing 'logging level netstack 4'. To the right of this area, there are two dropdown menus: 'Method' set to 'RESTCONF (Yang)' and 'Message format' set to 'json'. Below these, there's a 'POST' dropdown and a text input field containing 'restconf/data/Cisco-NX-OS-device:System/'. To the right of this field are three buttons: 'Send' (orange), 'Reset' (blue), and 'Convert' (blue). Below the input field, there's a tabbed interface with 'Request' selected, and other tabs for 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab shows a large empty text area with a 'Copy' button. Below the 'Request' tab, there's a 'Response' section with another large empty text area and a 'Copy' button.

**Step 2** Click **Message format** and choose either **json** or **xml**.

**Step 3** Enter a command in the text entry box in the top pane.

**Step 4** Choose a message format.

**Step 5** Click **Convert**.

**Example:**

For this example, the command is **logging level netstack 6** and the message format is json:

The screenshot shows the NX-API Sandbox interface. At the top, there's a header with the Cisco logo, "NX-API Sandbox", and links for "Quick Start", "Yang Documentation", "Yang Models", and "Logout". Below the header, there's a text input field containing the command "logging level netstack 6". To the right of this field, there are two dropdown menus: "Method:" set to "RESTCONF (Yang)" and "Message format:" set to "json". Below the input field, there's a "POST" dropdown and a text input field containing the URL "restconf/data/Cisco-NX-OS-device:System/". To the right of this URL field are three buttons: "Send" (orange), "Reset" (blue), and "Convert" (blue). Below the "Send" button, there are tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is selected, showing a JSON request body: 

```
{  "ipv4-items": {    "inst-items": {      "loggingLevel": "informational"    }  } }
```

 To the right of the JSON body is a green "Copy" button. Below the "Request" section, there's a "Response:" label and a large empty text area for the response. To the right of this area is another green "Copy" button.

**Example:**

For this example, the command is **logging level netstack 6** and the message format is xml:

The screenshot shows the NX-API Sandbox interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title. Navigation links include 'Quick Start', 'Yang Documentation', 'Yang Models', and 'Logout'. The main area is divided into two sections. The top section contains a text input field with the text 'logging level netstack 6'. To the right of this field, there are two dropdown menus: 'Method' set to 'RESTCONF (Yang)' and 'Message format' set to 'xml'. Below these is a 'POST' dropdown menu and a text input field containing 'restconf/data/Cisco-NX-OS-device/System/'. To the right of this field are three buttons: 'Send' (orange), 'Reset' (blue), and 'Convert' (blue). Below the input fields is a 'Request' tab, which is active. It shows an XML payload:
 

```
<ipv4-items>
  <inst-items>
    <loggingLevel>informational</loggingLevel>
  </inst-items>
</ipv4-items>
```

 To the right of the XML payload is a green 'Copy' button. Below the 'Request' tab is a 'Response' section, which is currently empty. To the right of the response area is another green 'Copy' button.

**Note**

When converting a negated CLI to a Yang payload using the XML or JSON message format, the sandbox throws a warning and disables the **Send** option. The warning message that appears depends on the message format:

- For the XML message format — "This is a Netconf payload as it is being generated for DELETE operation(s), hence SEND option is disabled for Restconf!"
- For the JSON message format—"This is a gRPC payload as it is being generated for DELETE operation(s), hence SEND option is disabled for Restconf!"

**Step 6**

You can also convert the request into the following formats by clicking on the appropriate tab in the **Request** pane:

- Python
- Python3
- Java
- JavaScript
- Go-Lang

**Note**

The Java-generated script does not work if you choose the PATCH option from the drop-down menu in the area above the Request tab. This is a known limitation with Java and is expected behavior.

