



XML Management Interface

- [XML Management Interface](#), on page 1

XML Management Interface

This chapter describes how to use the XML management interface to configure devices.

Feature History for XML Management Interface

The following table lists the release history for this feature:

Table 1: Feature History for XML Management Interface

| Feature Name | Releases | Feature Information |
|--------------------------|-------------|--|
| XML Management Interface | 7.3(0)D1(1) | Added the following NetConf enhancements: <ul style="list-style-type: none">• get-config• copy-config• validate• commit• lock• unlock• edit-config enhancements to support actions such as rollback on error, stop on error, continue on error, default operations, and candidate configuration. |

Information About the XML Management Interface

You can use the XML management interface to configure a device. The interface uses the XML-based Network Configuration Protocol (NETCONF), which allows you to manage devices and communicate over the interface with an XML management tool or program. The Cisco NX-OS implementation of NETCONF requires you to use a Secure Shell (SSH) session for communication with a device.

NETCONF is implemented with an XML Schema (XSD) that allows you to enclose device configuration elements within a remote procedure call (RPC) message. From within an RPC message, select one of the NETCONF operations that matches the type of command that you want the device to execute. You can configure the entire set of CLI commands on the device with NETCONF. For information about using NETCONF, see the [Creating NETCONF XML Instances, on page 6](#) and [RFC 4741](#).

For more information about using NETCONF over SSH, see [RFC 4742](#).

This section includes the following topics:

NETCONF Layers

The following table lists the NETCONF layers:

Table 2: NETCONF Layers

| Layer | Example |
|--------------------|---|
| Transport protocol | SSHv2 |
| RPC | rpc, rpc-reply |
| Operations | get-config, edit-config |
| Content | show or configuration command |

The following is a description of the four NETCONF layers:

- SSH transport protocol—Provides an encrypted connection between a client and the server.
- RPC tag—Introduces a configuration command from the requestor and the corresponding reply from the XML server.
- NETCONF operation tag—Indicates the type of configuration command.
- Content—Indicates the XML representation of the feature that you want to configure.

SSH xmlagent

The device software provides an SSH service called `xmlagent` that supports NETCONF over SSH Version 2.



Note The `xmlagent` service is referred to as the XML server in Cisco NX-OS software.

NETCONF over SSH is initiated by the exchange of a Hello message between the client and the XML server. After the initial exchange, the client sends XML requests, which the server responds to with XML responses. The client and server terminate requests and responses with the character sequence `>`. Because this character sequence is not valid in XML, the client and the server can interpret when messages end, which keeps communication synchronized.

The XML schemas that define the XML configuration instances that you can use are described in [Creating NETCONF XML Instances](#), on page 6.

Licensing Requirements for the XML Management Interface

| Product | License Requirement |
|-------------|---|
| Cisco NX-OS | The XML management interface requires no license. Any feature that is not included in a license package is bundled with the Cisco NX-OS image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the <i>Cisco NX-OS Licensing Guide</i> . |

Prerequisites to Using the XML Management Interface

Using the XML management interface has the following prerequisites:

- You must install SSHv2 on the client PC.
- You must install an XML management tool that supports NETCONF over SSH on the client PC.
- You must set the appropriate options for the XML server on the device.

Using the XML Management Interface

This section describes how to manually configure and use the XML management interface.



Note Use the XML management interface with the default settings on the device.

Configuring the SSH and the XML Server Options Through the CLI

By default, the SSH server is enabled on your device. If you disable SSH, you must enable it before you start an SSH session on the client PC.

You can configure the XML server options to control the number of concurrent sessions and the timeout for active sessions. You can also enable XML document validation and terminate XML sessions.



Note The XML server timeout applies only to active sessions.

For more information about configuring SSH, see http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/nx-os/security/configuration/guide/b_Cisco_Nexus_7000_NX-OS_Security_Configuration_Guide_Release_7-x.html.

For more information about the XML commands, see http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus7000/sw/system-management/guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide-RI.html.

-
- Step 1** Enter global configuration mode.
configure terminal
- Step 2** (Optional) Display information about XML server settings and active XML server sessions. You can find session numbers in the command output.
show xml server status
- Step 3** Validate XML documents for the specified server session.
xml server validate all
- Step 4** Terminate the specified XML server session.
xml server terminate *session*
- Step 5** (Optional) Disable the SSH server so that you can generate keys.
no feature ssh
- Step 6** Enable the SSH server. (The default is enabled.)
feature ssh
- Step 7** (Optional) Display the status of the SSH server.
show ssh server
- Step 8** Set the number of XML server sessions allowed.
xml server max-session *sessions*
The range is from 1 to 8. The default is 8.
- Step 9** Set the number of seconds after which an XML server session is terminated.
xml server timeout *seconds*
The range is from 1 to 1200. The default is 1200 seconds.
- Step 10** (Optional) Display information about the XML server settings and active XML server sessions.
show xml server status
- Step 11** (Optional) Saves the running configuration to the startup configuration.
copy running-config startup-config
-

Example

The following example shows how to configure SSH and XML server options through the CLI:

```
switch# configure terminal
switch(config)# xml server validate all
switch(config)# xml server terminate 8665
switch(config)# no feature ssh
switch(config)# feature ssh server
```

```
switch(config)# xml server max-session 6
switch(config)# xml server timeout 2400
switch(config)# copy running-config startup-config
```

Starting an SSHv2 Session

You can start an SSHv2 session on a client PC with the **ssh2** command that is similar to the following:

```
ssh2 username@ip-address -s xmlagent
```

Enter the login username, the IP address of the device, and the service to connect to. The `xmlagent` service is referred to as the XML server in the device software.



Note The SSH command syntax might differ based on the SSH software on the client PC.

If you do not receive a Hello message from the XML server, verify the following conditions:

- The SSH server is enabled on the device.
- The XML server's *max-sessions* option is adequate to support the number of SSH connections to the device.
- The active XML server sessions on the device are not all in use.

Sending a Hello Message

You must advertise your capabilities to the server with a Hello message before the server processes any other requests. When you start an SSH session to the XML server, the server responds immediately with a Hello message that informs the client of the server's capabilities. The XML server supports only base capabilities and in turn expects support only for these base capabilities from the client.

The following are sample Hello messages from the server and the client:



Note You must end all XML documents with `]]>]]>` to support synchronization in NETCONF over SSH.

Hello Message from a Server

```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
  <session-id>25241</session-id>
</hello>]]>]]>
```

Hello Message from a Client

```
<?xml version="1.0"?>
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
    <nc:capability>urn:ietf:params:xml:ns:netconf:base:1.0</nc:capability>
  </nc:capabilities>
</nc:hello>
```

```
</nc:capabilities>
</nc:hello>]]>]]>
```

Obtaining XML Schema Definition (XSD) Files

-
- Step 1** From your browser, navigate to the Cisco software download site at:
<http://software.cisco.com/download/navigator.html>
 The Download Software window is displayed.
- Step 2** From the list of products displayed, choose **Switches > Data Center Switches > platform model**.
- Step 3** If you are not already logged in as a registered Cisco user, you are prompted to log in now.
- Step 4** From the **Select a Software Type** list, choose **NX-OS XML Schema Definition**.
- Step 5** Find the desired release and click **Download**.
- Step 6** If you are requested to, follow the instructions to apply for eligibility to download strong encryption software images.
 The Cisco End User License Agreement is displayed.
- Step 7** Click **Agree** and follow the instructions to download the file to your PC.
-

Sending an XML Document to the XML Server

To send an XML document to the XML server through an SSH session that you opened in a command shell, copy the XML text from an editor and paste it into the SSH session. Although typically you use an automated method to send XML documents to the XML server, you can verify the SSH connection to the XML server through this copy-paste method.

The following are the guidelines to follow when sending an XML document to the XML server:

- Verify that the XML server has sent the Hello message immediately after you started the SSH session, by looking for the Hello message text in the command shell output.
- Send the client Hello message before you send XML requests. Note that the XML server sends the Hello response immediately, and no additional response is sent after you send the client Hello message.
- Always terminate the XML document with the character sequence `]]>]]>`.

Creating NETCONF XML Instances

You can create NETCONF XML instances by enclosing the XML device elements within an RPC tag and NETCONF operation tags. The XML device elements are defined in feature-based XML schema definition (XSD) files, which enclose available CLI commands in an XML format.

The following are the tags used in the NETCONF XML request in a framework context. Tag lines are marked with the following letter codes:

- X—XML declaration
- R—RPC request tag
- N—NETCONF operation tags

- D—Device tags

NETCONF XML Framework Context

```
X <?xml version="1.0"?>
R <nc:rpc message-id="1" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
R xmlns="http://www.cisco.com/nxos:1.0:nfcli">
N <nc:get>
N <nc:filter type="subtree">
D <show>
D <xml>
D <server>
D <status/>
D </server>
D </xml>
D </show>
N </nc:filter>
N </nc:get>
R </nc:rpc>]]>]]>
```



Note You must use your own XML editor or XML management interface tool to create XML instances.

RPC Request Tag

All NETCONF XML instances must begin with the RPC request tag `<rpc>`. The `<rpc>` element has a message ID (message-id) attribute. This message-id attribute is replicated in the `<rpc-reply>` and can be used to correlate requests and replies. The `<rpc>` node also contains the following XML namespace declarations:

- NETCONF namespace declaration—The `<rpc>` and NETCONF tags that are defined in the `urn:ietf:params:xml:ns:netconf:base:1.0` namespace, are present in the `netconf.xsd` schema file.
- Device namespace declaration—Device tags encapsulated by the `<rpc>` and NETCONF tags are defined in other namespaces. Device namespaces are feature oriented. Cisco NX-OS feature tags are defined in different namespaces. RPC Request Tag `<rpc>` is an example that uses the NFCLI feature. It declares that the device namespace is `xmlns=http://www.cisco.com/nxos:1.0:nfcli`. `nfcli.xsd` contains this namespace definition. For more information, see [Obtaining XML Schema Definition \(XSD\) Files, on page 6](#).

Examples

RPC Request Tag `<rpc>`

```
<nc:rpc message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
...
</nc:rpc>]]>]]>
```

Configuration Request

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
```

```

<nc:edit-config>
  <nc:target>
    <nc:running/>
  </nc:target>
  <nc:config>
    <configure>
      <__XML_MODE__exec_configure>
        <interface>
          <ethernet>
            <interface>2/30</interface>
            <__XML_MODE_if-ethernet>
              <__XML_MODE_if-eth-base>
                <description>
                  <desc_line>Marketing Network</desc_line>
                </description>
              </__XML_MODE_if-eth-base>
            </__XML_MODE_if-ethernet>
          </ethernet>
        </interface>
      </__XML_MODE__exec_configure>
    </configure>
  </nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>

```



Note `__XML__MODE` tags are used internally by the NETCONF agent. Some tags are present only as children of a certain `__XML__MODE`. By examining the schema file, you should be able to find the correct mode tag that leads to the tags representing the CLI command in XML.

NETCONF Operations Tags

NETCONF provides the following configuration operations:

Table 3: NETCONF Operations in Cisco NX-OS

| NETCONF Operation | Description | Example |
|-------------------|--|---|
| close-session | Closes the current XML server session. | NETCONF Close Session Instance, on page 17 |
| commit | Sets the running configuration to the current contents of candidate configuration. | NETCONF Commit Instance: Candidate Configuration Capability, on page 21 |
| confirmed-commit | Provides the parameters to commit the configuration for a specified period of time. If this operation is not followed up by a commit operation within the confirm-timeout period, the configuration will be reverted to the state prior to the confirmed-commit operation. | NETCONF Confirmed Commit Instance, on page 22 |
| copy-config | Copies the contents of the source configuration datastore to the target datastore. | NETCONF Copy Config Instance, on page 17 |

| NETCONF Operation | Description | Example |
|-------------------|---|---|
| delete-config | Operation not supported. | — |
| edit-config | Configures the features in the running configuration of the device. You use this operation for configuration commands. From Release 7.3(0)D1(1), support is added for the create, delete, and merge, rollback-on-error, continue-on-error, and stop-on-error actions. | NETCONF Edit Config Instance, on page 18 NETCONF Rollback-On-Error Instance, on page 22 |
| get | Receives configuration information from a device. You use this operation for show commands. The source of the data is the running configuration. | Creating NETCONF XML Instances, on page 6 |
| get-config | Retrieves all or part of a configuration. | NETCONF Get Config Instance, on page 20 |
| kill-session | Closes the specified XML server session. Note that you cannot close your own session. | NETCONF Kill Session Instance, on page 17 |
| lock | Allows a client to lock the configuration system of a device. | NETCONF Lock Instance, on page 20 |
| unlock | Releases the configuration lock issued by the session. | NETCONF Unlock Instance, on page 21 |
| validate | Checks a candidate's configuration for syntactical and semantic errors before applying the configuration to a device. | NETCONF Validate Capability Instance, on page 23 |

Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See [Obtaining XML Schema Definition \(XSD\) Files, on page 6](#).

Using this schema, it is possible to build an XML instance. The relevant portions of the nfcli.xsd schema file that was used to build the NETCONF instances. See ([Creating NETCONF XML Instances, on page 6](#)).

show xml Device Tags

```
<xs:element name="show" type="show_type_Cmd_show_xml"/>
<xs:complexType name="show_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>to display xml agent information</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice maxOccurs="1">
      <xs:element name="xml" minOccurs="1" type="xml_type_Cmd_show_xml"/>
      <xs:element name="debug" minOccurs="1" type="debug_type_Cmd_show_debug"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="xpath-filter" type="xs:string"/>

```

```
<xs:attribute name="uses-namespace" type="nxos:bool_true"/>
</xs:complexType>
```

Server Status Device Tags

```
<xs:complexType name="xml_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent server</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

Device Tag Response

```
<xs:complexType name="status_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="__XML_OPT_Cmd_show_xml__readonly__" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:group ref="og_Cmd_show_xml__readonly__" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="og_Cmd_show_xml__readonly__">
<xs:sequence>
<xs:element name="__readonly__" minOccurs="1" type="__readonly__type_Cmd_show_xml"/>
</xs:sequence>
</xs:group>
<xs:complexType name="__readonly__type_Cmd_show_xml">
<xs:sequence>
<xs:group ref="bg_Cmd_show_xml_operational_status" maxOccurs="1"/>
<xs:group ref="bg_Cmd_show_xml_maximum_sessions_configured" maxOccurs="1"/>
<xs:group ref="og_Cmd_show_xml_TABLE_sessions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
```



Note The `__XML_OPT_Cmd_show_xml__readonly__` tag is optional. This tag represents the response. For more information on responses, see [RPC Response Tag, on page 15](#).

You can use the | XML option to find the tags that you should use to execute a <get> operation. The following is an example of the | XML option. This shows you that the namespace-defining tag used to execute operations

on this device is `http://www.cisco.com/nxos:1.0:nfcli` and that the `nfcli.xsd` file can be used to build requests.

You can enclose the NETCONF operation tags and the device tags within the RPC tag. The `</rpc>` end tag is followed by the XML termination character sequence.

XML Example

```
Switch#> show xml server status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:nfcli">
<nf:data>
<show>
<xml>
<server>
<status>
<__XML__OPT_Cmd_show_xml__readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>
<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML__OPT_Cmd_show_xml__readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Extended NETCONF Operations

Cisco NX-OS supports an `<rpc>` operation named `<exec-command>`. The operation allows client applications to send CLI **configuration** and **show** commands and to receive responses to those commands as XML tags.

The following is an example of the tags used to configure an interface. Tag lines are marked with the following letter codes:

- X—XML declaration
- R—RPC request tag
- EO—Extended operation

The following table provides a detailed explanation of the operation tags:

Table 4: Operation Tags

| Tag | Description |
|-----------------------------------|-------------------------|
| <code><exec-command></code> | Executes a CLI command. |

| Tag | Description |
|-------|--|
| <cmd> | Contains the CLI command. A command can be a show command or configuration command. Multiple configuration commands should be separated by a semicolon (;). Although multiple show commands are not supported. You can send multiple configuration commands in different <cmd> tags as part of the same request. For more information, see the Example on <i>Configuration CLI Commands Sent Through <exec-command></i> . |

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

- <nf:ok>—All **configuration** commands are executed successfully.
- <nf:rpc-error>—Some commands have failed. The operation stops at the first error, and the <nf:rpc-error> subtree provides more information about which configuration has failed. Note that configurations executed before the failed command would have been applied to the running configuration.

Configuration CLI Commands Sent Through the <exec-command>

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

```
X <?xml version="1.0"?>
R <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
EO <nxos:exec-command>
EO <nxos:cmd>conf t ; interface ethernet 2/1 </nxos:cmd>
EO <nxos:cmd>channel-group 2000 ; no shut; </nxos:cmd>
EO </nxos:exec-command>
R </nf:rpc>]]]]>
```

Response to CLI Commands Sent Through the <exec-command>

The following is the response to a send operation:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]]]>
```

Show CLI Commands Sent Through the <exec-command>

The following example shows how the **show** CLI commands that are sent through the <exec-command> can be used to retrieve data:

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show interface brief</nxos:cmd>
```

```
</nxos:exec-command>
</nf:rpc>]]>]]>
```

Response to the show CLI Commands Sent Through the <exec-command>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0"
xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
<nf:data>
<mod:show>
<mod:interface>
<mod:__XML_OPT_Cmd_show_interface_brief__readonly__>
<mod:__readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
<mod:ip_addr>172.23.152.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML_OPT_Cmd_show_interface_brief__readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Failed Configuration

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 1.1.1.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
```

```

<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

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

The `<rpc-reply>` as a result of a **show** command that is sent through the `<cmd>` tag contains the XML output of the **show** command.

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

Combination of configure and show Commands

```

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

```

show CLI Commands Sent Through the <exec-command>

```

<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>

```

```

<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

NETCONF Replies

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

RPC Response Tag

The following example shows the RPC response tag `<rpc-reply>`:

RPC Response Tag `<rpc-reply>`

```

<nc:rpc-reply message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
<ok/>
</nc:rpc-reply>]]>]]>

```

RPC Response Elements

The elements `<ok>`, `<data>`, and `<rpc-error>` can appear in the RPC response. The following table describes the RPC response elements that can appear in the `<rpc-reply>` tag:

Table 5: RPC Response Elements

| Element | Description |
|--------------------------------|--|
| <code><ok></code> | The RPC request completed successfully. This element is used when no data is returned in the response. |
| <code><data></code> | The RPC request completed successfully. The data associated with the RPC request is enclosed in the <code><data></code> element. |
| <code><rpc-error></code> | The RPC request failed. Error information is enclosed in the <code><rpc-error></code> element. |

Interpreting the Tags Encapsulated in the data Tag

The device tags encapsulated in the `<data>` tag contain the request, followed by the response. A client application can safely ignore all the tags before the `<readonly>` tag, as show in the following example:

RPC Reply Data

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>

```

```

<show>
<interface>
<__XML__OPT_Cmd_show_interface_brief__readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief__readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```



Note <__XML__OPT.*> and <__XML__BLK.*> appear in responses and are sometimes used in requests. These tags are used by the NETCONF agent and are present in responses after the <__readonly__> tag. They are necessary in requests, and should be added according to the schema file to reach the XML tag that represents the CLI command.

Example XML Instances

This section provides examples of the following XML instances:

- [NETCONF Close Session Instance, on page 17](#)
- [NETCONF Kill Session Instance, on page 17](#)
- [NETCONF Copy Config Instance, on page 17](#)
- [NETCONF Edit Config Instance, on page 18](#)
- [NETCONF Get Config Instance, on page 20](#)
- [NETCONF Lock Instance, on page 20](#)
- [NETCONF Unlock Instance, on page 21](#)
- [NETCONF Commit Instance: Candidate Configuration Capability, on page 21](#)
- [NETCONF Confirmed Commit Instance, on page 22](#)
- [NETCONF Rollback-On-Error Instance, on page 22](#)
- [NETCONF Validate Capability Instance, on page 23](#)

NETCONF Close Session Instance

The following examples show the close-session request, followed by the close-session response:

Close Session Request

```
<?xml version="1.0"?>
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:close-session/>
</nc:rpc>]]>]]>
```

Close Session Response

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF Kill Session Instance

The following examples show the kill session request, followed by the kill session response:

Kill Session Request

```
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:kill-session>
<nc:session-id>25241</nc:session-id>
</nc:kill-session>
</nc:rpc>]]>]]>
```

Kill Session Response

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF Copy Config Instance

The following examples show the copy config request, followed by the copy config response:

Copy Config Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
<target>
<running/>
</target>
<source>
```

```
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

Copy Config Response

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```



Note <startup/> is not supported as a source or target datastore.

To perform any copy operation on startup-config like “copy running-config startup-config”, you need to fallback to the <exec-command> method.

NETCONF Edit Config Instance

The following examples show the use of NETCONF edit config:

Edit Config Request

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nc:edit-config>
<nc:target>
<nc:running/>
</nc:target>
<nc:config>
<configure>
<__XML__MODE__exec_configure>
<interface>
<ethernet>
<interface>2/30</interface>
<__XML__MODE_if-ethernet>
<__XML__MODE_if-eth-base>
<description>
<desc_line>Marketing Network</desc_line>
</description>
</__XML__MODE_if-eth-base>
</__XML__MODE_if-ethernet>
</ethernet>
</interface>
</__XML__MODE__exec_configure>
</configure>
</nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>
```

Edit Config Response

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager" message-id="16">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

The operation attribute in edit config identifies the point in configuration where the specified operation will be performed. If the operation attribute is not specified, the configuration is merged into the existing configuration data store. The operation attribute can have the following values:

- create
- merge
- delete

Edit Config: Delete Operation Request

The following example shows how to delete the configuration of interface Ethernet 0/0 from the running configuration:

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<default-operation>none</default-operation>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<top xmlns="http://example.com/schema/1.2/config">
<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

Response to Edit Config: Delete Operation

The following example shows how to edit the configuration of interface Ethernet 0/0 from the running configuration:

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```



Note XML edit-config with candidate datastore is not supported with 1.0 version xml request. It is supported only with the newer version, which can be generated using xmlin tool.

NETCONF Get Config Instance

The following examples show the use of NETCONF get config:

Get Config Request to Retrieve the Entire Subtree

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<users/>
</top>
</filter>
</get-config>
</rpc>]]>]]>
```

Get Config Response with Results of a Query

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>
```

NETCONF Lock Instance

The following examples show a lock request, a success response, and a response to an unsuccessful attempt:

Lock Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<lock>
<target>
<running/>
</target>
</lock>
</rpc>]]>]]>
```

Response to a Successful Acquisition of Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/> <!-- lock succeeded -->
</rpc-reply>]]>]]>
```

Response to an Unsuccessful Attempt to Acquire Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error> <!-- lock failed -->
<error-type>protocol</error-type>
<error-tag>lock-denied</error-tag>
<error-severity>error</error-severity>
<error-message>
Lock failed, lock is already held
</error-message>
<error-info>
<session-id>454</session-id>
<!-- lock is held by NETCONF session 454 -->
</error-info>
</rpc-error>
</rpc-reply>]]>]]>
```

NETCONF Unlock Instance

The following examples show the use of NETCONF unlock:

Unlock Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<unlock>
<target>
<running/>
</target>
</unlock>
</rpc>
```

Response to an Unlock Request

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF Commit Instance: Candidate Configuration Capability

The following examples show a commit operation and a commit reply:

Commit Operation

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
```

Commit Reply

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF Confirmed Commit Instance

The following examples show a confirmed commit operation and a confirmed commit reply:

Confirmed Commit Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit>
<confirmed/>
<confirm-timeout>120</confirm-timeout>
</commit>
</rpc>]]]]>
```

Confirmed Commit Response

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]]]>
```

NETCONF Rollback-On-Error Instance

The following examples show the how to configure rollback on error and the response to this request:

Rollback-On-Error Capability

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<error-option>rollback-on-error</error-option>
<config>
<top xmlns="http://example.com/schema/1.2/config">
<interface>
<name>Ethernet0/0</name>
<mtu>100000</mtu>
</interface>
```

```

</top>
</config>
</edit-config>
</rpc>]]>]]>

```

Rollback-On-Error Response

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>

```

NETCONF Validate Capability Instance

The following examples show the use of NETCONF validate capability that is identified by the string `urn:ietf:params:netconf:capability:validate:1.0`:

Validate Request

```

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>

```

Response to Validate Request

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>

```

Additional References

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

RFCs

| RFCs | Title |
|--------------------------|--|
| RFC 4741 | NETCONF Configuration Protocol |
| RFC 4742 | Using the NETCONF Configuration Protocol over Secure Shell (SSH) |

