

NETCONF Agent

This chapter contains the following topics:

- About the NETCONF Agent, on page 1
- Guidelines and Limitations for NETCONF, on page 2
- Configuring the NETCONF Agent, on page 3
- Establishing a NETCONF Session, on page 5
- NETCONF Read and Write Configuration, on page 6
- NETCONF Execution, on page 14
- NETCONF Notifications, on page 17
- NETCONF Examples, on page 21
- Troubleshooting the NETCONF Agent, on page 25

About the NETCONF Agent

The Network Configuration Protocol (NETCONF) is a network management protocol defined by RFC 6241. Cisco NX-OS provides a NETCONF agent which is a client-facing interface that provides secure transport over SSH for the client requests and server responses in the form of a YANG model, encoded in XML.

NETCONF defines configuration datastores and a set of Create, Read, Update, and Delete (CRUD) operations that allow manipulation and query on these datastores. Three datastores are supported on NX-OS: running, startup, and candidate. Here's a brief descriptions of the operations that are supported:

Table 1: Supported Operations

Operation	Description
get	Retrieve running configuration and operational state
get-config	Retrieve configuration from specified datastore
edit-config	Load specified configuration to the specified target datastore
close-session	Request graceful termination of a session
kill-session	Force the termination of a session

Operation	Description
copy-config	Create or replace datastore with the contents of another datastore
lock	Lock the datastore
unlock	Unlock the datastore
validate	Validate the contents of the specified configuration
commit	Commit the candidate configuration as the new current running configuration
cancel-commit	Cancel an ongoing confirmed commit
discard-changes	Revert the candidate configuration to the current running configuration

Guidelines and Limitations for NETCONF

The NETCONF Agent has the following guideline and limitation:

- Cisco NX-OS supports both the Cisco Device YANG model and OpenConfig models in NETCONF notifications.
- The device YANG model defines ephemeral data and they are marked with a comment "// Ephemeral data". These nonpersistent large-volume data is handled differently from the rest of the model. They are returned only when `<get>` query's `<filter>` parameter points specifically to the particular element marked with the comment. Refer to the ephemeral data support documentation for detailed information on the usage.
- Beginning with Cisco NX-OS Release 9.3(3), NETCONF is RFC 6241 compliant with the following exceptions:
  - Sibling content match nodes are logically combined in an "OR" expression instead of an "AND" expression. (Section 6.2.5)
  - Once a candidate datastore has been edited, the running configuration for the same property must not be edited.
- In a single Get request, the number of objects that are supported is 250,000. If you see the following error, it means that the data requested is more than 250,000. To avoid this error, send requests with filters querying for a narrower scope of data.

  ```
  too many objects(459134 > 250000) to query the entire device model.
  ```
- NETCONF does not support enhanced Role-Based Access Control (RBAC) as specified in RFC 6536. Only users with a "network-admin" role are granted access to the NETCONF agent.
- The `<edit-config>` "replace" operation sometimes might not work due to run-time default values and behaviors that are implemented by the affected system component. Therefore, it's better to base the configuration to replace on the configuration obtained through the `<get-config>` query instead of the NX-API Developer Sandbox.

- The Cisco NX-OS NETCONF server supports a maximum of five subscriptions, one subscription per client session.
- Per RFC 5277, autonomous notifications support NETCONF, SYSLOG, and SNMP streams for event sources. In this release, Cisco NX-OS supports NETCONF streams only.
- Cisco NX-OS does not support the Replay option for subscriptions. Because Start Time and Stop Time options are part of Replay, they are not supported.
- For a stream subscription and filtering, support is only for subtree filtering. XPath filtering is not supported.
- When the Cisco NX-OS NETCONF Agent is operating under a heavy load, it is possible that some event notifications can get dropped.
-
- You cannot configure NETCONF on a switch if NAT is already configured. The configurations of NETCONF and NAT are incompatible and cannot coexist on the same switch.
- Cisco NX-OS supports both the Cisco Device YANG model and OpenConfig models. Support for OpenConfig models in NETCONF notifications begins with the Cisco NX-OS 9.3(5) release.
- Beginning with 10.2(1)F release, the operations checkpoint, rollback, install, import ca certificate, module reload, individual module reload, and copy file are supported.
- When openconfig-acl NETCONF GET operations are performed with L2 MAC leaf property value(s) as input, it is recommended to input the letters in MAC addresses in the uppercase format (AA:AA:AA:AA:AA:AA). For example, source-mac: 0A:0B:0C:0D:0E:0F.

Configuring the NETCONF Agent

Configuring the NETCONF Agent Over SSH for Cisco NX-OS 9.3(5) and Later

This procedure describes how to enable and configure the NETCONF Agent over SSH.



Note Use this procedure with Cisco NX-OS Release 9.3(5) and later.

Before you begin

Before communicating with the switch using NETCONF, the NETCONF Agent must be enabled. The NETCONF Agent is enabled or disabled by entering the **[no] feature netconf** command.

SUMMARY STEPS

1. **configure terminal**
2. **feature netconf**
3. (Optional) **netconf idle-timeout** *it-num*
4. (Optional) **netconf sessions** *num-sessions*

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `switch# configure terminal`	Enters global configuration mode.
Step 2	**feature netconf** **Example:** `switch(config)# feature netconf`	Enable NETCONF services.
Step 3	(Optional) **netconf idle-timeout** *it-num* **Example:** `switch(config)# netconf idle-timeout 5`	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The range of*it-num* is 0-1440 minutes. The default timeout is 5 minutes. A value of 0 disables timeout.
Step 4	(Optional) **netconf sessions** *num-sessions* **Example:** `switch(config)# netconf sessions 5`	Specifies the number of maximum simultaneous client sessions. The range of *num-sessions* is 1-10. The default is 5 sessions.

Configuring the NETCONF Agent for Cisco NX-OS 9.3(4) and Earlier

Note Use this procedure with Cisco NX-OS Release 9.3(4) and earlier.

The NETCONF Agent supports the following optional configuration parameters under the `[netconf]` section in the configuration file (`/etc/mtx.conf`).

Parameter	Description
idle_timeout	(Optional) Specifies the timeout in minutes after which idle client sessions are disconnected. The default value is 5 minutes. A value of 0 disables timeout.
limit	(Optional) Specifies the number of maximum simultaneous client sessions. The default value is 5 sessions. The range is 1-10.

The following is an example of the `[netconf]` section in the configuration file:

```
[netconf]
```

```
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

For the modified configuration file to take effect, you must restart the NETCONF Agent using the CLI command [**no**] **feature netconf** to disable and reenable.

Establishing a NETCONF Session

NETCONF is a connection-oriented protocol requiring a persistent connection between client and server. The NETCONF agent on the switch listens at port 830 of the management port IP address. The client can establish a connection with the NETCONF subsystem over SSH. When a client establishes a session with the NETCONF agent, the server sends a `<hello>` message to the client. The client likewise must send its `<hello>` message to the server. The `<hello>` messages are exchanged simultaneously as soon as the connection is open. Each `<hello>` message contains a list of the sending peer's protocol version and capabilities. These messages are used to determine protocol compatibility and capabilities. Both NETCONF peers must verify that a common protocol version is advertised by the other peer's `<hello>` message. Also, the server's `<hello>` message must include a `<session-id>` whereas the client's `<hello>` message must not.

The following shows an example session establishment using the **ssh** command. The first `<hello>` message is received from the server and the second message is sent from the client. The server's `<hello>` message shows the protocol version "urn:ietf:params:netconf:base:1.1" and NETCONF base capabilities that are supported on Cisco NX-OS Release 9.3(4). Also, the server's `<hello>` message includes supported data models. They might not match the models supported in the current Cisco NX-OS release.



Note The server's `<hello>` message has a `<session-id>`, but the client's message does not.

```
client-host % ssh admin@172.19.193.166 -p 830 -s netconf
User Access Verification
Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>

<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=report-all</capability>


<capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2020-04-20&amp;amp;module=Cisco-NX-OS-device</capability>


<capability>http://openconfig.net/yang/acl?revision=2019-11-27&amp;amp;module=openconfig-acl&amp;amp;deviations=cisco-nx-openconfig-acl-deviations</capability>


<capability>http://openconfig.net/yang/bfd?revision=2019-10-25&amp;amp;module=openconfig-bfd&amp;amp;deviations=cisco-nx-openconfig-bfd-deviations</capability>
```

```
        ...
    </capabilities>
    <session-id>1286775422</session-id>
</hello>
]]>]]><hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
```

Using NETCONF with the **ssh** command is not convenient and is prone to error, as the complexity for message framing can be seen from RFC 6242 (Using the NETCONF Protocol over SSH). The **ssh** command is used for the example above for illustration purposes only. There are various clients written for NETCONF which are recommended over the **ssh** command. The ncclient is one such example and is used in the Usage Examples section.

NETCONF supports two operations for terminating a session, namely, `<close-session>` and `<kill-session>`. When the server receives a `<close-session>` request, it gracefully terminates the session by releasing any locks and resources associated with the session and closing the connection with the client. The following is an example of the `<close-session>` request and response for success:

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>

<rpc-reply message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

The `<kill-session>` request forces the termination of another session and requires `<session-id>` in the request message. Upon receiving the `<kill-session>` request, the server terminates current operations, releases locks and resources, and closes the connection associated with the specified session ID. The following is an example of the `<kill-session>` request and response for success:

```
<rpc message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>296324181</session-id>
  </kill-session>
</rpc>

<rpc-reply message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Besides the `<close-session>` and `<kill-session>` requests, a session is terminated automatically if the client does send any request for a certain length of time. The default is five minutes. See Configuring the NETCONF Agent for configuring the idle timeout.

NETCONF Read and Write Configuration

This section describes supported base protocol operations to manipulate and query datastores. The client can send RPC messages for these operations after establishing a session with the NETCONF agent. Basic usage explanations are given and RFC 6242 can be referred to for thorough details about these operations.

<get-config>

This operation retrieves configuration data from a specified datastore. The supported parameters are `<source>` and `<filter>`. The `<source>` specifies the datastore being queried such as `<running/>`, which holds the currently active configuration. The `<filter>` specifies the portions of the specified datastore to retrieve.

The following are examples of `<get-config>` request and response messages.

- Retrieve the entire `<System>` subtree:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        ...
        </System>
    </data>
</rpc-reply>
```

- Retrieve a specific list item:

```
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            ...
                            <rtctrl-items>
                                <enforceFirstAs>enabled</enforceFirstAs>
                                <fibAccelerate>disabled</fibAccelerate>
```

```
                                    <logNeighborChanges>enabled</logNeighborChanges>
                                    <supprRt>enabled</supprRt>
                                </rtctrl-items>
                                <rtrId>1.2.3.4</rtrId>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </data>
    </rpc-reply>
```

<edit-config>

This operation writes a specified configuration to the target datastore. The <target> parameter specifies the datastore being edited, such as <running/> or <candidate/>. The candidate datastore can be manipulated without impacting the running datastore until its changes are committed. For more information, see the <commit> section. The `<config>` parameter specifies the modeled data to be written to the target datastore. The model is specified by the "xmlns" attribute. Any number of elements in the `<config>` subtree may contain an "operation" attribute. The operation of an element is inherited by its descendent elements until it's overridden by a new "operation" attribute. The supported operations are "merge", "replace", "create", "delete", and "remove". The "remove" operation is different from "delete" in that no error is returned if the configuration data does not exist. If the "operation" attribute is not specified, the merge operation is assumed as default; the default operation can be overridden by the optional `<default-operation>` parameter, which has "merge", "replace" or "none".

The following are examples of `<edit-config>` request and response messages.

- Create a port-channel named "po5" with MTU 9216 and the description in the running configuration:

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="create">
                            <id>po5</id>
                            <mtu>9216</mtu>
                            <descr>port-channel 5</descr>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
    <ok/>
</rpc-reply>
```

- Replace all configurations of a port-channel with new configurations:

```
<rpc message-id="104" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
```

```
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
               <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="replace">
                            <id>po5</id>
                            <mtu>1500</mtu>
                            <adminSt>down</adminSt>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="104">
    <ok/>
</rpc-reply>
```

- Delete a port-channel:

```
<rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
           <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="delete">
                            <id>po5</id>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="105">
    <ok/>
</rpc-reply>
```

<copy-config>

This operation replaces the target configuration datastore with the contents of source configuration datastore. The parameters for source datastore and target datastore are `<source>` and `<target>`, respectively.

The following are examples of `<copy-config>` request and response messages.

- Copy from running configuration to startup configuration:

```
<rpc message-id="106" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <startup/>
        </target>
        <source>
```

```
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="106">
    <ok/>
</rpc-reply>
```

- Copy from running configuration to candidate configuration:

```
<rpc message-id="107" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <candidate/>
        </target>
        <source>
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="107">
    <ok/>
</rpc-reply>
```

<lock>

The `<lock>` operation allows a client to lock the configuration datastore, preventing other clients from locking or modifying the datastore. The lock that is held by the client is released with either the `<unlock>` operation or termination of a session. The `<target>` parameter is used to specify the datastore to be locked.

The following are examples of `<lock>` request and response messages.

- A successful acquisition of a lock:

```
<rpc message-id="108" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <running/>
        </target>
    </lock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="108">
    <ok/>
</rpc-reply>
```

- A failed attempt to acquire a lock already in use by another session:

```
<rpc message-id="109" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <candidate/>
        </target>
    </lock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="109">
    <rpc-error>
        <error-type>protocol</error-type>
```

```
        <error-tag>lock-denied</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">Lock failed, lock is already held</error-message>

        <error-info>
            <session-id>1553704357</session-id>
        </error-info>
    </rpc-error>
</rpc-reply>
```

<unlock>

The `<unlock>` operation releases a configuration lock, obtained with the <lock> operation. Only the same session that issued the `<lock>` operation can use the `<unlock>` operation. The `<target>` parameter is used to specify the datastore to be unlocked.

The following is an example of `<unlock>` request and response messages.

- Unlock

```
<rpc message-id="110" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <unlock>
        <target>
            <candidate/>
        </target>
    </unlock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="110">
    <ok/>
</rpc-reply>
```

<get>

The <get> operation retrieves running configuration and operational state data. The supported parameter is `<filter>`. The `<filter>` specifies the portions of the running configuration operational state data to retrieve.

The following is an example of `<get>` request and response messages.

- Retrieve running configuration and operational state data of a list item:

```
<rpc message-id="111" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="111">
```

```
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            <always>disabled</always>
                            <bestPathIntvl>300</bestPathIntvl>
                            <clusterId>120</clusterId>
                          <firstPeerUpTs>2020-04-20T16:19:03.784+00:00</firstPeerUpTs>

                            <holdIntvl>180</holdIntvl>
                            <id>1</id>
                            <kaIntvl>60</kaIntvl>
                            <mode>fabric</mode>
                            <numEstPeers>0</numEstPeers>
                            <numPeers>0</numPeers>
                            <numPeersPending>0</numPeersPending>
                            <operRtrId>1.2.3.4</operRtrId>
                            <operSt>up</operSt>
                            <pfxPeerTimeout>90</pfxPeerTimeout>
                            <pfxPeerWaitTime>90</pfxPeerWaitTime>
                            <reConnIntvl>60</reConnIntvl>
                            <rtrId>1.2.3.4</rtrId>
                            <vnid>0</vnid>
                            ...
                        </Dom-list>
                    </dom-items>
                </inst-items>
            </bgp-items>
        </System>
    </data>
</rpc-reply>
```

<validate>

This operation validates the configuration contents of the candidate datastore. It is useful for validating the configuration changes made on the candidate datastore before committing them to the running datastore. The `<source>` parameter supports `<candidate/>`.

The following is an example of `<validate>` request and response messages.

- Validate the contents of the candidate datastore:

```
<rpc message-id="112" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <validate>
        <source>
            <candidate/>
        </source>
    </validate>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="112">
    <ok/>
</rpc-reply>
```

<commit>

This operation commits the candidate configuration to the running configuration. The operation without any parameter is considered final and cannot be reverted. If `<commit>` is issued with the `<confirmed/>` parameter, it is considered a confirmed commit, and commit is finalized only if it is followed by another <commit> operation without the `<confirmed/>` parameter. That is, the confirming commit. The confirmed commit allows two parameters: <confirm-timeout> and <persist>. The `<confirm-timeout>` is the period in seconds before the confirmed commit is reverted, restoring the running configuration to its state before the confirmed commit was issued, unless the confirming commit is issued before or the timeout is reset by another confirmed commit. If the `<confirm-timeout>` is not specified, the default timeout is 600 seconds. Also, the confirmed commit is reverted if the session is terminated. The `<persist>` parameter makes the confirmed commit to persist even if the session is terminated. The value of the `<persist>` parameter is used to identify the confirmed commit from any session, and must be used as the value of the `<persist-id>` parameter of subsequent confirmed commit or confirming commit.

The following are examples of `<commit>` request and response messages.

- Commit the contents of the candidate datastore:

```
<rpc message-id="113" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="113">
    <ok/>
</rpc-reply>
```

- Confirmed commit with the timeout:

```
<rpc message-id="114" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
        <confirm-timeout>120</confirm-timeout>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="114">
    <ok/>
</rpc-reply>
```

- Start a persistent confirmed commit and then confirm the persistent confirmed commit:

```
<rpc message-id="115" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
        <persist>ID1234</persist>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="115">
    <ok/>
</rpc-reply>

<!-- confirm the persistent confirmed-commit, from the same session or another session
 -->
<rpc message-id="116" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <persist-id>ID1234</persist-id>
    </commit>
</rpc>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="116">
    <ok/>
</rpc-reply>
```

<cancel-commit>

This operation cancels an ongoing confirmed commit. If a confirmed commit from a different session needs to be canceled, the `<persist-id>` parameter must be used with the same value that was given in the `<persist>` parameter of the confirmed commit.

- Cancel the confirmed commit from the same sessions:

```
<rpc message-id="117" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <cancel-commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="117">
    <ok/>
</rpc-reply>
```

<discard-changes>

This operation discards any uncommitted changes that are made on the candidate configuration by resetting back to the content of the running configuration. No parameter is required.

The following is an example of `<discard-changes>` request and response messages.

- Discard the changes made on the candidate datastore:

```
<rpc message-id="118" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <discard-changes/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="118">
    <ok/>
</rpc-reply>
```

NETCONF Execution

About Model Driven Operations in NETCONF

Table 2: About Model Driven Operations in NETCONF

Operation	NETCONF RPC	CLI
Checkpoint	checkpoint	checkpoint <name> checkpoint <file>

Operation	NETCONF RPC	CLI
Rollback	rollback	rollback running-config checkpoint <name> rollback running-config checkpoint <file>
Install	install_all_nxos install_add install_activate install_deactivate install_commit install_remove	install all nxos <image> install {add | activate | deactivate | commit | remove} <rpm>
Import Crypto Certificate	import_ca_certificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>
Switch Reload or Module Reload	reload	reload [timer <seconds>] reload module <module number>
Copy File	copy	copy <source> <destination>

Model Driven Operations Examples

Model Driven Operations Examples

Creating checkpoint using filename option:

```
RPC:
<rpc message-id="checkpoint-3" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <file>bootflash:my_checkpoint2</file>
  </checkpoint>
</rpc>
```

Creating checkpoint uisng checkpoint name, description:

```
RPC:
<rpc message-id="checkpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>create</action>
      <name>my_checkpoint1</name>
      <description>test checkpoint one</description>
  </checkpoint>
</rpc>
```

Deleting checkpoint using checkpoint name:

```
RPC:
<rpc message-id="delatecheckpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>delete</action>
      <name>my_checkpoint1</name>
  </checkpoint>
</rpc>
```

Rollback:



Note The following option tags can be used as atomic, stop-at-first-failure, best-effort.

```
<rpc message-id="rollback-cfg-option1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <name>my_checkpoint1</name>
    <option>atomic</option>
</rollback>
</rpc>
```

Rollback using file option

```
<rpc message-id="rollback-cfg1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
     <file>bootflash:my_checkpoint2</file>
</rollback>
</rpc>
```

Copy file

Copy any file from remote server to switch storage(example: bootflash)

For Kerry tftp: protocol supports for file transfer.

```
<rpc message-id="copy-file-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <source>tftp://172.27.xxx.xxx//<file_location?/tls1-server.pfx</source>
    <destination>bootflash:</destination>
      <vrf>management</vrf>
</copy>
</rpc>
```

Import CA Certificate

Pre-requisite: my_truspoint should be already created on the switch:

```
<rpc message-id="import_ca_certificate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<import_ca_certificate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <trustpoint>my_trustpoint</trustpoint>
    <pkcs12>tls1-server.pfx</pkcs12>
    <passphrase>xxxxxx</passphrase>
</import_ca_certificate>
</rpc>
```

Install RPM package EXEC RPC commands

Install <add>

```
<rpc message-id="install-add-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_add xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
 <add>rpm_packagenamehere_from_bootflash</add>
</install_add>
</rpc>
```

Install <activate>

```
<rpc message-id="install-activate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_activate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <activate> rpm_packagenamehere_from_bootflash</activate>
</install_activate>
</rpc>
```

Install <deactivate>

```
<rpc message-id="install-deactivate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_deactivate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <deactivate>rpm_packagenamehere_from_bootflash </deactivate>
 </install_deactivate>
</rpc>
```

Install <remove>

```
<rpc message-id="rpc-install_remove-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_remove xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
<remove>rpm_packagenamehere_from_bootflash </remove>
</install_remove>
</rpc>
```

Install all nx-os image

```
<rpc message-id="rpc-install_all_nxos-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<install_all_nxos xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <nxos>nxos.image.bin.upg</nxos>
</install_all_nxos>
</rpc>
```

Reload module number

```
<rpc message-id="reload-module-pyld1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <module>29</module>
</reload>
</rpc>
```

Reload



Note When Client requests or sends the following RPC, the exec command executes switch reload and further Netconf client does not receive <ok> response.

```
<rpc message-id="563" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
</rpc>
```

NETCONF Notifications

About NETCONF Notifications

NETCONF notification is a mechanism where a NETCONF client can subscribe to system events and then receive notifications to these events from a NETCONF agent. These features are defined in RFC 5277. Beginning with Cisco NX-OS Release 9.3(1), support for NETCONF notifications began as described in RFC 5277. This is an optional capability that is advertised in the NETCONF hello message.

A NETCONF client can subscribe for notifications using Deviceyang or OpenConfig models. Support for OpenConfig models in NETCONF notifications begins with Cisco NX-OS Release 9.3(5).

With this support, any NETCONF client can:

- Subscribe to event notifications.

Each subscription is a one-time request over a session from a NETCONF client. The Cisco NX-OS NETCONF agent responds, and the subscription is active until the session is explicitly closed by the NETCONF client. The subscription can also be closed by an administrative action, such as a switch restart or disabling NETCONF feature on the switch. The subscription is active as long as the underlying NETCONF session is active. The events that are generated for these subscribed filters are sent as notifications to the client. Clients can subscribe to notifications for system events. For example, port state change, fan speed change, and process memory change to name a few. Also, configuration events such as a new feature being enabled.

- Receive event notifications.

  An event notification is a well-formed XML document that contains information about the configuration or operational events on the switch. The NETCONF client can send filtering criteria in the subscription request to specify a subset of events instead of all events.

- Interleave event notifications with other operations.

  The Cisco NX-OS NETCONF agent can receive, process, and respond to NETCONF requests on a session with an active notification subscription.

Capabilities Exchange

During the NETCONF handshake, the Cisco NX-OS NETCONF server sends the <capabilities> element to the connecting NETCONF clients to indicate what requests that the server can process. As part of the exchange, the server includes the following identifiers, which inform the client that the Cisco NX-OS NETCONF server supports both notifications and interleave.

Capability identifier for notification:

```
urn:ietf:params:netconf:capability:notification:1.0
```

Capability identifier for interleave:

```
urn:ietf:params:netconf:capability:interleave:1.0
```

Event Stream Discovery

The client can discover the Cisco NX-OS NETCONF server's supported streams by using a NETCONF <get> operation for all available <streams>. Cisco NX-OS supports the NETCONF stream only. Discovering event streams occurs through a request and reply sequence.

Request to retrieve available streams:

Any NETCONF client can send a NETCONF <get> request with filter for <streams> to identify all supported streams. The following example shows the payload of a client request message:

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get>
      <filter type="subtree">
        <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
           <streams/>
        </netconf>
      </filter>
     </get>
```

```
</rpc>
```

Reply:

The Cisco NX-OS NETCONF server replies with all the event streams that are available and to which the client can subscribe. Cisco NX-OS supports the NETCONF stream only.

```
<rpc-reply message-id="101"
                    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <data>
       <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
        <streams>
           <stream>
              <name>NETCONF</name>
              <description>default NETCONF event stream </description>
           </stream>
        </streams>
       </netconf>
     </data>
</rpc-reply>
```

Creating Subscriptions

NETCONF clients can create subscriptions for events on the switch through an RPC with a <create-subscription> protocol operation. When the Cisco NX-OS NETCONF server responds with the `<ok/>` element, the subscription is active.

Unlike synchronous Get and Set operations, a subscription is a persistent, asynchronous operation. The subscription stays active until the client explicitly closes the subscription or the session goes offline. For example, by a switch restart.

If a client subscribes to event notifications, but it goes offline, the server terminates the subscription and closes the session.

If a subscription is closed, the NETCONF client must reconnect and create the subscription again to receive all event notifications.

The server does not initiate subscriptions, so you must to write client programs that contain the `<create-subscription>` operation. The following is an example for `<create-subscription>` sent by any NETCONF client:

```
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>NETCONF</stream>
    <filter xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0" type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                          <operSt/>
                        </phys-items>
                    </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
      </filter>
```

```
</create-subscription>
```

The <create-subscription> operation supports any of the following options:

- <stream>, which specifies which stream of events the client wants to subscribe to. If you specify no stream, by default, events in the NETCONF stream are sent to the client.
- <filter>, which enables filtering the events to provide a subset of events carried on the stream.

The Cisco NX-OS NETCONF server responds back with an <ok> message if the server is able to create the subscription successfully.

The following is a sample successful response received in the client for the <create-subscription> request that it sent to the server.

Response for <create-subscription>, received in the client:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:6ff0bda6-d3f1-4288-9a7e-0f30581e4bab">
    <ok/>
</rpc-reply>
```



Note Subscriptions with Replay are not supported, so the Start Time and Stop Time options cannot be used.

Receiving Notifications

When the NETCONF client has successfully created a subscription, the Cisco NX-OS NETCONF server begins sending relevant event notifications, for any events in the switch, for the filter used. The event notification is its own XML-formatted document that contains the notification element.

The following is a sample notification for an Ethernet interface going down, when the client subscribed to interface operSt, from the DeviceYang model. The <create-subscription> is in the Creating Subscriptions section.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2020-05-05T10:22:52.260+00:00</eventTime>
    <operation>modified</operation>
    <event>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                            <operSt>down</operSt>
                        </phys-items>
                    </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </event>
</notification>
```

The <notification> messages contain the following fields:

- <eventTime>, the date and timestamp of when the event occurred.
- <operation>, the type of event on the model node.
- <event>, the model data to which the client is subscribed.

Terminating Subscriptions

Subscriptions are terminated when the NETCONF client sends specific operations to the Cisco NX-OS NETCONF server in the payload of a NETCONF message. Subscription termination occurs in any of the following ways:

- Closing the subscription session, which occurs when the <close-session> operation is sent to the NETCONF Server for a specific subscription session.
- Terminating the NETCONF session, which occurs when the <kill-session> operation is sent to the NETCONF server.

Every subscription is tied to one NETCONF session. It is a one-to-one relationship.

NETCONF Examples



Note All examples in this section use the ncclient python library.

Connecting Cisco NX-OS with the ncclient

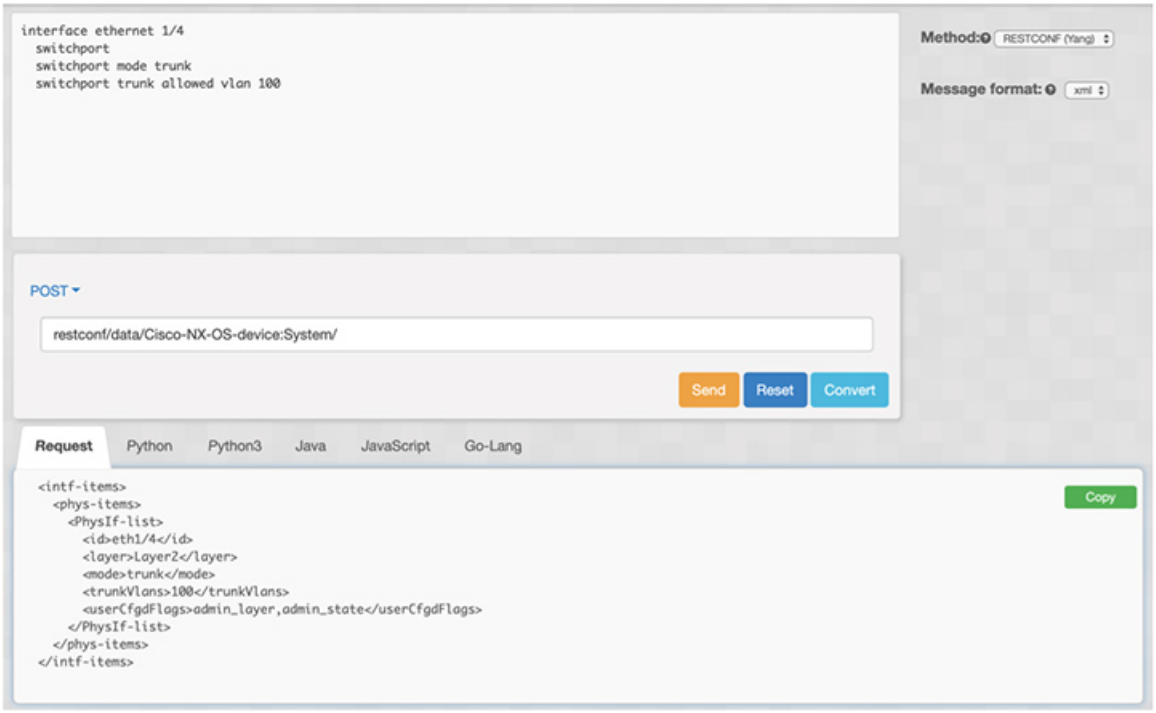
The ncclient is a Python library for NETCONF clients. The following is an example of how to establish a connection to Cisco NX-OS from the ncclient Manager API:

```
device = {
    "address": "10.10.10.10",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}
with manager.connect(host = device["address"],
                         port = device["netconf_port"],
                         username = device["username"],
                         password = device["password"],
                         hostkey_verify = False) as m:
    # do your stuff
```

Using the Sandbox to Generate the NETCONF Payload

Refer to NXAPI Developer Sandbox section to enable it. In order to generate a payload for NETCONF, change the method to RESTCONF (Yang) and message format to XML. Enter the command you need to convert in the text window, click **Convert** and the equivalent payload is displayed in the Request text box:

Figure 1: NCCLIENT



Getting Configuration Data from Cisco NX-OS

Here is an example of how to use the ncclient to get the BGP configuration from Cisco NX-OS:

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco!"
}


# create a main() method
def main():
    bgp_dom = """
    <filter type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list/>
                        </dom-items>
                    </inst-items>
                </bgp-items>
        </System>
    </filter>
    """

    with manager.connect(host=device["address"],
```

```
                        port=device["netconf_port"],
                        username=device["username"],
                        password=device["password"],
                        hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get_config(source='running', filter=bgp_dom)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))


if __name__ == '__main__':
    sys.exit(main())
```

Getting the Running Configuration and Operational Data from Cisco NX-OS

Here is example of getting the interface counters of all the physical interfaces on Cisco NX-OS:

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():

    intf_ctr_filter = """
    <filter>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                <PhysIf-list>
                    <dbgIfIn-items/>
                    <dbgIfOut-items/>
                </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </filter>"""

    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get(filter=intf_ctr_filter)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))

if __name__ == '__main__':
    sys.exit(main())
```

Creating a New Configuration Using NETCONF

Here is example of how to create VLAN 100 with name using edit config of ncclient:

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
    add_vlan = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bd-items>
          <bd-items>
            <BD-list>
              <fabEncap>vlan-100</fabEncap>
              <name>inb_mgmt</name>
            </BD-list>
          </bd-items>
        </bd-items>
      </System>
    </config>
    """


    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=add_vlan)
        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

Deleting Configuration Using NETCONF

Here is example of deleting a loopback interface from Cisco NX-OS:

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
```

```
    remove_loopback = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <lb-items>
            <LbRtdIf-list operation="delete">
              <id>lo10</id>
            </LbRtdIf-list>
          </lb-items>
        </intf-items>
      </System>
    </config>"""


    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=remove_loopback)

        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

Troubleshooting the NETCONF Agent

Troubleshooting Connectivity

- From a client system, ping the management port of the switch to verify that the switch is reachable.
- In Cisco NX-OS, enter the **show feature | inc netconf** command to check the agent status.
- There is the XML Management Interface (also known as xmlagent), which is quite different from and often confused as the NETCONF Agent. Please ensure that you connect to the correct port 830 and receive a correct <hello> message (similar to what is shown in the Establishing a NETCONF Session section) from the server if the server does not respond with the correct NETCONF messages.