

NETCONF Agent

- About the NETCONF Agent, on page 1
- Revision History, on page 2
- Guidelines and Limitations for NETCONF, on page 2
- Configuring the NETCONF Agent, on page 4
- Manage a NETCONF Session, on page 7
- NETCONF Get / Set, on page 10
- NETCONF Notifications, on page 19
- NETCONF Device YANG RPC, on page 22
- Netconf Client Examples, on page 26
- Troubleshooting the NETCONF Agent, on page 30
- About NETCONF Explicit Mode, on page 33
- Guidelines and Limitations, on page 33
- NETCONF Explicit Mode Get/Set, on page 34
- Add Configuration Using CLI, on page 38

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 or TLS 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 description 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
create-subscription	Subscribe to event notification stream

Revision History

Release	Description
9.3(1)	Beginning with NX-OS 9.3(1), NETCONF get and get-config requests from the NETCONF client to the switch must contain an explicit namespace and filter
9.3(3)	RFC 6241 Compliant
9.3(5)	Support for OpenConfig models in NETCONF notifications
10.2(1)	Support for RPC operations: checkpoint, rollback, install, import ca certificate, module reload, individual module reload, and copy file
10.3(1)	Support Cisco Nexus 9800 platform switches
10.3(2)	Support RBAC for edit-config
10.3(3)	Support Netconf transport over TLS
10.4(3)	Support Netconf explicit mode
10.5(3)	Support Netconf parallel execution

Guidelines and Limitations for NETCONF

The NETCONF Agent has the following guideline and limitation:

Netconf YANG Operations

- 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.

- Cisco NX-OS NETCONF does not fully support enhanced Role-Based Access Control (RBAC) as specified in RFC 6536. Instead, it allows to grant per path EDIT permission to non **network-admin** roles.
- Users without **network-admin** access, are not permitted to access DME structures.
- NETCONF `get` and `get-config` requests from the NETCONF client to the switch must contain an explicit namespace and filter. This requirement affects requests to the OpenConfig YANG and NETCONF Device models. If you see an error response that is like the following, the requests are not carrying a namespace:

  ```
  Request without namespace and filter is an unsupported operation
  ```

  The following example shows a proper request and response with the correct behavior in NX-OS release 9.3(1) and later.

  ```
  <get>
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      </System>
    </filter>
  </get>
  ```

- 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.
- 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
  ```

Native Device RPC

- The Cisco NX-OS supports Device YANG RPC operations: checkpoint, rollback, install, import ca certificate, module reload, individual module reloads, and copy file are supported.

Subscription / Notification

- The Cisco NX-OS 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.
- **Netconf Parallel Execution**

- Cisco NX-OS supports a maximum of ten parallel Netconf requests, one request per client session.

- Beginning with Cisco NX-OS Release 10.4(3)F, NETCONF is supported on 92348GC-X.

Transport Layer Security (TLS)

- Beginning with Cisco NX-OS Release 10.4(3)F, TLS v1.3 is supported, and the minimally supported version is v1.2.

Netconf Parallel Execution

- Beginning with Cisco NX-OS Release 10.5(3)F, a maximum of ten parallel Netconf requests (one request per client session) is supported.

Configuring the NETCONF Agent

Configuring the NETCONF Agent Over SSH

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

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*

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`	Enables NETCONF services.
Step 3	(Optional) **netconf idle-timeout** *it-num* **Example:**	Specifies the timeout in minutes after which idle client sessions are disconnected. The range of *it-num* is 0-1440

	Command or Action	Purpose
	`switch(config)# netconf idle-timeout 5`	minutes. The default timeout is 5 minutes. A value of 0 disables timeout.

Configuring the NETCONF Agent Over TLS

This procedure describes how to enable and configure the NETCONF Agent over TLS. This enables an additional TLS transport channel to run side-by-side together with the SSH transport.

Before you begin

Prepare and sign the required certificate files for both the server and client-side authentication.

Note This is not specific to Netconf, so you can re-use the existing trustpoint files.

SUMMARY STEPS

1. **configure terminal**
2. (Optional) **crypto ca trustpoint** *server-trustpoint*
3. (Optional) **crypto ca trustpoint** *server-trustpoint* **pkcs12 bootflash:** *server-ca-file pkcs-password*
4. (Optional) **crypto ca trustpoint** *client-root-trustpoint*
5. (Optional) **rsakeypair** *client-key*
6. (Optional) **crypto ca authenticate** *client-root-trustpoint*
7. **netconf tls certificate** *server-trustpoint*
8. **netconf tls client root certificate** *client-root-trustpoint*
9. **netconf tls port** *port*

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `switch# configure terminal`	Enters global configuration mode.
Step 2	(Optional) **crypto ca trustpoint** *server-trustpoint* **Example:** `switch(config)# crypto ca trustpoint tls_server_trustpoint`	Creates a trustpoint for server authentication **Note** The Steps 2-6 is optional if you already have an usable server and client trustpoints.
Step 3	(Optional) **crypto ca trustpoint** *server-trustpoint* **pkcs12 bootflash:** *server-ca-file pkcs-password*	Imports the server pkcs12 file to the trustpoint.

	Command or Action	Purpose
	Example: `switch(config)# crypto ca import tls_server_trustpoint pkcs12 bootflash:server.pfx test`	
Step 4	(Optional) **crypto ca trustpoint** *client-root-trustpoint* **Example:** `switch(config)# crypto ca trustpoint tls_client_trustpoint`	Creates a trustpoint for client authentication.
Step 5	(Optional) **rsakeypair** *client-key* **Example:** `switch(config)# rsakeypair client-key`	Generates a rsa key pair for the client trustpoint.
Step 6	(Optional) **crypto ca authenticate** *client-root-trustpoint* **Example:** `switch(config)# crypto ca authenticate tls_client_trustpoint`	Imports the client certificate. This step requires manual copy paste. Please follow the instruction<need info on which instruction?.>
Step 7	**netconf tls certificate** *server-trustpoint* **Example:** `switch(config)# netconf tls certificate tls_server_trustpoint`	Enters the trustpoint to host the server identity certificate.
Step 8	**netconf tls client root certificate** *client-root-trustpoint* **Example:** `switch(config)# netconf tls client root certificate tls_client_trustpoint`	Enters the trustpoint to host the client CA root certificate.
Step 9	**netconf tls port** *port* **Example:** `switch(config)# netconf tls port 7000`	Specify the TLS port. The default port is 6513.

Generating Key/Certificate Example

The following is an example for generating a self-signed key/certificate in the switch bash shell.

Note that this is only for experimental usage. For more information on generating identity certificates, see the **Installing Identity Certificates** section of the *Cisco Nexus 9000 Series NX-OS Security Configuration Guide*.

Note This task is an example of how a certificate can be generated on a NX-OS switch. You can also generate a certificate in any Linux environment. In a production environment, the user should consider using a CA signed certificate.

1. Generate the self-signed key and pem files.

```
switch# run bash sudo su
bash-4.3# openssl req -x509 -newkey rsa:2048 -keyout self_sign2048.key -out
self_sign2048.pem -days 365 -nodes
```

2. After generating the key and pem files, you must bundle the key and pem files for use in the trustpoint CA Association.

```
switch# run bash sudo su bash-4.3# cd /bootflash/
bash-4.3# openssl pkcs12 -export -out self_sign2048.pfx -inkey self_sign2048.key -in
self_sign2048.pem -certfile self_sign2048.pem -password pass:Ciscolab123!
bash-4.3# exit
```

3. Set up the trustpoint CA Association by inputting in the pkcs12 bundle into the trustpoint.

```
switch(config)# crypto ca trustpoint mytrustpoint
switch(config-trustpoint)# crypto ca import mytrustpoint pkcs12 self_sign2048.pfx
Ciscolab123!
```

4. Verify the setup.

```
Verify the setup.
switch(config)# show crypto ca certificates Trustpoint: mytrustpoint certificate:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R serial=0413
notBefore=Nov 5 16:48:58 2015 GMT notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
CA certificate 0: subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R serial=0413
notBefore=Nov 5 16:48:58 2015 GMT notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
```

Manage 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. Note that NETCONF agent is often confused with the XML server running on port 22. For the XML server running on port 22, refer XML Management Interface chapter of the *Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 10.5(x)*.

Start Session

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 client and server 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 include the <session-id>.

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 includes the supported data models. This below example might not reflect 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;module=Cisco-NX-OS-device

    </capability>


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

    </capability>


<capability>http://openconfig.net/yang/bfd?revision=2019-10-25&amp;module=openconfig-bfd&amp;deviations=cisco-nx-openconfig-bfd-deviations</capability>
...
    </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 raw **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 above **ssh** command is used for illustration purposes only. There exist various clients written for NETCONF which are recommended over the **ssh** command. The ncclient is one such example. Please refer to the *Examples* section.

Terminate Session

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 not 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 Get / Set

This section describes supported NETCONF 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 specified configuration to the target datastore.

The <target> parameter specifies the datastore being edited, such as <running/> or <candidate/>. The <running/> datastore is manipulated to immediately change the switch config while 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 intended 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”. 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 could be “merge”, “replace” or “none”.

- The “merge” operation is different from “create” in that no error is returned if the config data already exists.
- The “remove” operation is different from “delete” in that no error is returned if the config data does not exist.

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 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. This is an optional capability that is advertised in the NETCONF hello message.

A NETCONF client can subscribe for notifications using Device YANG or OpenConfig models.

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 if 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 config or operational change events. For example, port state change, fan speed change, process memory change, feature enabled, to name a few.

- **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>
```

Create 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 the user must write client programs that contain the send the `<create-subscription>` operation.

The following is an example for `<create-subscription>` sent by a 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 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 can create the subscription successfully.

The following is a sample successful response received in the client for the `<create-subscription>`:

```
<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 that “Start Time” and “Stop Time” options cannot be used.

Receive 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 matching the requested filter. The event notification is its own XML-formatted document that contains the notification element.

The following is a sample notification when an interface “eth1/54/1” went down. In this case, the client has subscribed to interface operSt with the example from the above example.

```
<?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.

Terminate Subscriptions

Subscriptions are terminated when the NETCONF client sends specific request to the Cisco NX-OS NETCONF server 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 Device YANG RPC

About Model Driven RPC in NETCONF

In addition to manipulate the switch configuration, YANG offers the extensibility to trigger specific actions in the switch via YANG model, which is equivalent to invoke EXEC mode commands through CLI. Cisco NX-OS has defined the following Native Device RPC.

Operation	Device YANG RPC	CLI	
Checkpoint	checkpoint	**checkpoint** <*name*> **checkpoint** <*file*>	180

Operation	Device YANG RPC	CLI	
Rollback	rollback	**rollback running-config checkpoint** <*name*> **rollback running-config checkpoint** <*file*>	6900
Install	install_all_nxos	**install all nxos** <*image*>	1800
	install_add install_activate install_deactivate install_commit install_remove	**install** {**add** | **activate** | **deactivate** | **commit** | **remove**} <*rpm*>	180
Import Crypto Certificate	import_ca_certificate	**crypto ca import** <*trustpoint*> **pkcs12** <*file*> <*passphrase*>	5
Switch Reload or Module Reload	reload	**reload** [**timer** <*seconds*>] **reload module** <*module number*>	180
Copy File	copy	**copy** <*source*> <*destination*>	7200
File/Directory Information	dir	dir <*file*>	180
Run cli	cli	<cli command>	1800

Native Device RPC Examples

- **Creating checkpoint using filename option**

```
<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 using checkpoint name, description**

```
<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
  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 options 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)

```
<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>
```

Directory or file information

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

Run CLI command on switch

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

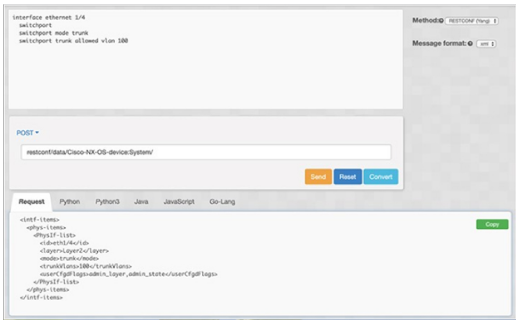
Netconf Client Examples

Using the Sandbox to Generate the NETCONF Payload

Refer to NXAPI Developer Sandbox section to enable it.

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: NX-OS Developer Sandbox



Connecting Cisco NX-OS with the ncclient

Note There exist various NETCONF clients. All examples in this section use the particular ncclient python library

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
```

Getting Configuration Data

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

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

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

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

Check Feature Status

- In Cisco NX-OS, enter the **show feature | inc netconf** command to check the agent config.
- To view the status of the NETCONF agent, use the **show feature** command.

```
switch-1# show feature | grep netconf
restconf 1 enabled
switch-1#
```

Check Connectivity

- From a client system, ping the management port of the switch to verify that the switch is reachable.
- 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 (like what is shown in the Establishing a NETCONF Session section) from the server if the server does not respond with the correct NETCONF messages.

Check TLS

If TLS is used, the user can check the TLS status using the **show netconf internal tls service statistics** command.

First check the server start time, which tells whether the TLS server is running.

If the server is not running, then further check the Cert and the Client Root Cert to see whether the certificate is valid.

```
# show netconf internal tls service statistics
=============
TLS Service
=============
Port           : 6513
Cert notBefore : Nov  5 16:48:58 2015 GMT
Cert notAfter  : Nov  5 16:48:58 2035 GMT
Client Root Cert notBefore : Oct  1 18:00:24 2020 GMT
Client Root Cert notAfter  : Sep 26 18:00:24 2040 GMT
Server restarts            : 108
Created sessions           : 54
Start                      : 02/23 01:13:10
Run time                   : 142 hour(s) 15 min(s)sec(s)
```

If the server appears to run properly, further check the session status using the **show netconf internal tls session all summary** command:

- If the server does not receive the TLC connection at all, then would recommend debugging the client-side connectivity.
- If the server indeed receives the session but rejects the session immediately, then would recommend checking the user authentication. Check whether the certification is configured properly.

```
# show netconf internal tls session all summary
N9k (config)# show netconf internal tls session all summary
=============
TLS Session
=============
* - history
Client                                          Read (KB)   Write (KB)  Status
----------------------------------------------- ----------- ----------- ------------
*10.28.23.116:35490                                    0.4         1.1 End
*10.28.23.116:35494                                    0.4         1.1 End
--                                                     0.0         0.0 Listening
```

Accounting Log for NETCONF Agent

For write operations such as <edit-config>, <commit> or <abort>, NETCONF would emit corresponding accounting log. It would include both the original received request, as well as the eventual changes applied to the switch. This is useful to check the history of config changes via Netconf.

You can see the accounting log using the **show accounting log** command.

Refer the following NETCONF request as an example:

```
---
<edit-config>
<config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
 <intf-items>
  <lb-items>
   <LbRtdIf-list>
    <id>lo10</id>
    <descr nc:operation="create">test</descr>
   </LbRtdIf-list>
  </lb-items>
 </intf-items>
</System>
</config>
</edit-config>
---
```

The accounting log shall include the following items:

- Changes applied to the switch:

Item	Description
Context	Session ID and user
Operation	COMMIT/ABORT
Database	Running or Candidate
ConfigMO	MO tree’s text representation. Up to 3K characters.
Status	SUCCESS/FAILED

Example:

```
Wed Jun 29 13:48:03
2022:type=update:id=2496515744:user=admin:cmd=(COMMIT),database=[running],configMo=[
<topSystem childAction="" dn="sys" status="created,modified"><interfaceEntity
childAction="" rn="intf"
```

```
status="created,modified"><l3LbRtdIf childAction="" id="lo10" rn="lb-[lo10]"
status="created,modified"/></interfaceEntity></topSystem>] (SUCCESS)
```

- Original received request

Item	Description
Context	Session ID and user
Operation	NETCONF:EDIT-CONFIG, NETCONF:COMMIT, NETCONF
Source IP	NETCONF Client IP
Payload	Received XML Request. Up to 3K characters.
Status	SUCCESS/FAILED
tem	Description
Context	Session ID and user

Example:

```
Wed Jun 29 13:48:03
2022:type=update:id=2496515744:user=admin:cmd=(NETCONF:EDITCONFIG),sourceIp=[192.168.1.2],
payload=[<edit-config><config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System
xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"><intf-items><lb-items><LbRtdIflist><id>lo10</id>
<descr
nc:operation="remove">test</descr></LbRtdIf-list></lb-items></intfitems></System></config></edit-config>]

(SUCCESS)
```

In case of failed request, based on the failed scenarios, a user may not observe both the logs.

- Invalid request:

  The invalid request would be rejected without making a configuration change, thus only the original request would be logged.

  Example:

  ```
  Wed Jun 29 20:08:36
  2022:type=update:id=2517274784:user=admin:cmd=(NETCONF:EDITCONFIG),
  sourceIp=[192.168.1.2],payload=[<edit-config><config
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <System
  xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"><intf-items><lb-items><LbRtdIflist
   nc:operation="create"><id>lo10</id>
  </LbRtdIf-list></lb-items></intfitems></System></config></edit-config>] (FAILED)
  ```

- Failures due to various configuration restrictions:

  In this case, both the failed configuration attempt and the original request would be logged.

  Example:

  ```
  Wed Jun 29 20:11:04
  2022:type=update:id=2517274784:user=admin:cmd=(COMMIT),database=[running],
  configMo=[<topSystem childAction="" dn="sys"
  status="created,modified"><telemetryEntity
  childAction=""rn="tm" status="created,modified"><telemetryCertificate childAction=""

  filename="foo" hostname="foo" rn="certificate" status="created,modified"
  trustpoint="test"/>
  ```

```
</telemetryEntity></topSystem>] (FAILED)
Wed Jun 29 20:11:04
2022:type=update:id=2517274784:user=admin:cmd=(NETCONF:EDITCONFIG),sourceIp=[192.168.1.2],
payload=[<edit-config><config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
<tm-items><certificateitems><trustpoint>test</trustpoint><hostname>foo</hostname>
<filename>foo</filename></certificateitems></tm-items></System></config></edit-config>]
 (FAILED
```

About NETCONF Explicit Mode

The Network Configuration Protocol Explicit mode is a network management protocol defined by RFC 6243. This protocol defines three standard modes to read the default configuration from the NETCONF server. The standard modes are **report-all**, **trim**, and **explicit**.

Cisco NX-OS already supports **report-all** mode. When configuration is read using **report-all** mode, all the configuration including default configurations are visible to the users. This feature additionally allows NETCONF client to read configuration using the **explicit** mode. In explicit mode, Cisco NX-OS only exposes the configuration that is explicitly done by the user.

Cisco NX-OS only support the explicit mode with NETCONF **<get-config>** and **<edit-config>** RPC:

- **<get-config>**

  In explicit mode, **<get-config>** retrieves the data which is explicitly configured by the user irrespective of its value.

- **<edit-config>**

  In explicit mode, the operations such as create, delete, merge, replace and remove works slightly different from the **report-all** mode.

  - A valid **create** operation attribute for a data node that has been set by a client to its schema default value fails with a **data-exists** error-tag. A valid **create** operation attribute for a data node that has been set by the server to its schema default value succeeds.
  - A valid **delete** operation attribute for a data node that has been set by a client to its schema default value succeeds. A valid **delete** operation attribute for a data node that has been set by the server to its schema default value fails with a **data-missing** error-tag.

Guidelines and Limitations

- Cisco NX-OS does not support operations other than `get-config` and `edit-config`.
- As this feature partially supports RFC 6243, switch would not advertise the with-default explicit capability.
- The explicit get-config response from clean boot switch would never be empty.
- This feature does not guarantee the expected response when multiple switches are allowed to access the switch at the same time.
- Upgrade from prior 10.3(4) is subject to the below limitations:
  - install ... non-xx

- install ..

- Reload from any release is subject to the below limitation:
- reload ascii
- Upgrade from previous releases
- Downgrade from previous releases

NETCONF Explicit Mode Get/Set

To illustrate the explicit mode, consider the below telemetry configuration model.

```
+--rw tm-items
  |   +--rw dest-items
  |   |  +--rw DestGroup-list* [id]
  |   |       +--rw id                   telemetry_IDType
  |   |       +--rw addr-items
  |   |       |  +--rw Dest-list* [addr port]
  |   |       |     +--rw addr      address_Ip
  |   |       |     +--rw port      uint16
  |   |       |     +--rw proto?    telemetry_Protocol      <===== Default Config
  |   |       |     +--rw enc?       telemetry_Encoding    <===== Default Config
  |   |       |     +--rw nodeid?   telemetry_NodeIDorZero
```

Add config

- The following edit-config request would configure id along with the destination address and port

```
<edit-config>
    <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

    <target>
        <running/>
    </target>
    <config>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <tm-items>
                <dest-items>
                    <DestGroup-list>

            <id
              xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
              nc:operation="create">1</id>
                        <addr-items>
                            <Dest-list>
                                <addr>2.2.2.2</addr>
                                <port>2</port>
                            </Dest-list>
                        </addr-items>
                    </DestGroup-list>
                </dest-items>
            </tm-items>
        </System>
    </config>
</edit-config>
-----
```

```
Response:

-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <ok/>
</rpc-reply>
```

- This following report-all get-config request would only return all the data.

```
<get-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">report-all</with-defaults>

  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
      </tm-items>
    </System>
  </filter>
</get-config>

 Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <adminSt>enabled</adminSt>
        <batchDmeEvt>true</batchDmeEvt>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>GPB</enc>
                <proto>gRPC</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

- This following explicit get-config request would only return the configured data.

```
Explicit <get-config> request:

<get-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
```

```
  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items/>
    </System>
  </filter>
</get-config>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

Add Configuration Using CLI

Below are the example steps to configure NETCONF request/response in explicit mode.

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 2.2.2.2 port 2 protocol UDP encoding JSON
```

- The above explicit get-config request would return following response

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>JSON</enc>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
```

```
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

Remove configuration

```
<edit-config>
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

  <target>
    <running/>
  </target>
  <config>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc
                  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
                  nc:operation="delete">JSON</enc>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </config>
</edit-config>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <ok/>
</rpc-reply>
```

- The above explicit get-config request would return following response

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <proto>UDP</proto>
              </Dest-list>
```

```
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

Add Configuration Using CLI

Follow below steps to configure explicit mode:

SUMMARY STEPS

1. **configure terminal**
2. **telemetry**
3. **destination-group** *dgrp_id*
4. **ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `switch# configure terminal`	Enter the global configuration mode.
Step 2	**telemetry** **Example:** `switch# telemetry`	Enter configuration mode for streaming telemetry.
Step 3	**destination-group** *dgrp_id* **Example:** `switch# destination-group 1`	Create a destination group and enter destination group configuration mode.
Step 4	**ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol* **Example:** `switch# ip address 2.2.2.2 port 2 protocol UDP encoding JSON`	Specify an IPv4 IP address and port to receive encoded telemetry data.

Example

- The above explicit get-config request would return following response

```
-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>JSON</enc>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>

-----
```

Remove configuration

```
Request:

-----
<edit-config>
    <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

    <target>
        <running/>
    </target>
    <config>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <tm-items>
                <dest-items>
                    <DestGroup-list>
                        <id>1</id>
                        <addr-items>
                            <Dest-list>
                                <addr>2.2.2.2</addr>
                                <port>2</port>
                               <enc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
 nc:operation="delete">JSON</enc>
                            </Dest-list>
                        </addr-items>
                    </DestGroup-list>
                </dest-items>
            </tm-items>
        </System>
    </config>
</edit-config>

-----

Response:

-----
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>

-----
```

- The above explicit get-config request would return following response

```
-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>

-----
```