

Use NETCONF Protocol to Define Network Operations with Data Models

Table 1: Feature History Table

Feature Name	Release Information	Description
Unified NETCONF V1.0 and V1.1	Release 7.3.1	Cisco IOS XR supports NETCONF 1.0 and 1.1 programmable management interfaces. With this release, a client can choose to establish a NETCONF 1.0 or 1.1 session using a separate interface for both these formats. This enhancement provides a secure channel to operate the network with both interface specifications.

XR devices ship with the YANG files that define the data models they support. Using a management protocol such as NETCONF or gRPC, you can programmatically query a device for the list of models it supports and retrieve the model files.

Network Configuration Protocol (NETCONF) is a standard transport protocol that communicates with network devices. NETCONF provides mechanisms to edit configuration data and retrieve operational data from network devices. The configuration data represents the way interfaces, routing protocols and other network features are provisioned. The operational data represents the interface statistics, memory utilization, errors, and so on.

NETCONF uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. It uses a simple RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. The client can be a script or application that runs as part of a network manager. The server is a network device such as a router. NETCONF defines how to communicate with the devices, but does not handle what data is exchanged between the client and the server.

Note Accessing the router via NETCONF grants by default write permissions for a user, in spite of read-only access configured for this user for CLI access, as CLI authorization is bypassed.

NETCONF Session

A NETCONF session is the logical connection between a network configuration application (client) and a network device (router). The configuration attributes can be changed during any authorized session; the effects are visible in all sessions. NETCONF is connection-oriented, with SSH as the underlying transport. NETCONF sessions are established with a `hello` message, where features and capabilities are announced. At the end of each message, the NETCONF agent sends the `]]>]]>` marker. Sessions are terminated using `close` or `kill` messages.

Cisco IOS XR supports NETCONF 1.0 and 1.1 programmable management interfaces that are handled using two separate interfaces. From IOS XR, Release 7.3.1, a client can choose to establish a NETCONF 1.0 or 1.1 session using an interface for both these formats. A NETCONF proxy process waits for the `hello` message from its peer. If the proxy does not receive a `hello` message within the timeout period, it sends a NETCONF 1.1 `hello` message.

```
<?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:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability
--snip--
</capabilities>
<session-id>5</session-id>
</hello>]]>]]>
```

The following examples show the `hello` messages for the NETCONF versions:

```
netconf-xml agent listens on port 22

netconf-yang agent listens on port 830
```

Version 1.0 The NETCONF XML agent accepts the message.

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

Version 1.1 The NETCONF YANG agent accepts the message.

```
<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 1.1, the RPC requests begin with `#<number>` and end with `##`. The number indicates how many bytes that follow the request.

Example:

```
#371
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <filter>
    <isis xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-clns-isis-oper">
      <instances>
        <instance>
```

```
          <neighbors/>
          <instance-name/>
        </instance>
      </instances>
    </isis>
  </filter>
</get>
</rpc>

##
```

Configure NETCONF Agent

To configure a NETCONF TTY agent, use the **netconf agent tty** command. In this example, you configure the *throttle* and *session timeout* parameters:

```
netconf agent tty
         throttle (memory | process-rate)
         session timeout
```

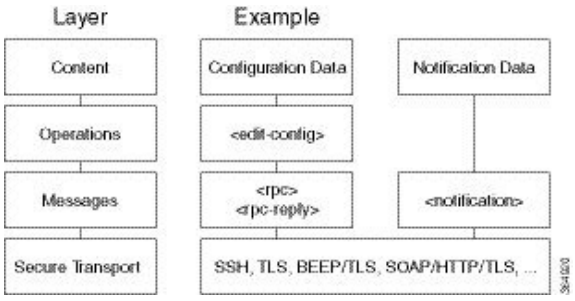
To enable the NETCONF SSH agent, use the following command:

```
ssh server v2
netconf-yang agent ssh
```

NETCONF Layers

NETCONF protocol can be partitioned into four layers:

Figure 1: NETCONF Layers



- **Content layer:** includes configuration and notification data
- **Operations layer:** defines a set of base protocol operations invoked as RPC methods with XML-encoded parameters
- **Messages layer:** provides a simple, transport-independent framing mechanism for encoding RPCs and notifications
- **Secure Transport layer:** provides a communication path between the client and the server

For more information about NETCONF, refer RFC 6241.

This article describes, with a use case to configure the local time on a router, how data models help in a faster programmatic configuration as compared to CLI.

- NETCONF Operations, on page 4
- Retrieve Default Parameters Using with-defaults Capability, on page 8
- Retrieve Transaction ID for NSO Operations, on page 14

• Set Router Clock Using Data Model in a NETCONF Session, on page 16

NETCONF Operations

NETCONF defines one or more configuration datastores and allows configuration operations on the datastores. A configuration datastore is a complete set of configuration data that is required to get a device from its initial default state into a desired operational state. The configuration datastore does not include state data or executive commands.

The base protocol includes the following NETCONF operations:

```
|  +--get-config
|  +--edit-Config
|     +--merge
|     +--replace
|     +--create
|     +--delete
|     +--remove
|     +--default-operations
|        +--merge
|        +--replace
|        +--none
|  +--get
|  +--lock
|  +--unLock
|  +--close-session
|  +--kill-session
```

These NETCONF operations are described in the following table:

NETCONF Operation	Description	Example
<get-config>	Retrieves all or part of a specified configuration from a named data store	Retrieve specific interface configuration details from running configuration using filter option `<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-config><source><running/></source><filter><interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"\><interface-configuration><active>act</active><interface-name>TenGigE0/0/0/2/0</interface-name></interface-configuration></interface-configurations></filter></get-config></rpc>`

NETCONF Operation	Description	Example
<get>	Retrieves running configuration and device state information	Retrieve all acl configuration and device state information. Request: <get> <filter> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/> </filter> </get>
<edit-config>	Loads all or part of a specified configuration to the specified target configuration	Configure ACL configs using **Merge** operation <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target><candidate/></target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-cfg" xc:operation=”merge”> <accesses> <access> <access-list-name>aclv4-1</access-list-name> <access-list-entries> <access-list-entry> <sequence-number>10</sequence-number> <remark>GUEST</remark> </access-list-entry> <access-list-entry> <sequence-number>20</sequence-number> <grant>permit</grant> <source-network> <source-address>172.0.0.0</source-address> <source-wild-card-bits>0.0.255.255</source-wild-card-bits> </source-network> </access-list-entry> </access-list-entries> </access> </accesses> </ipv4-acl-and-prefix-list> </config> </edit-config> </rpc> Commit: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <commit/> </rpc>

NETCONF Operation	Description	Example

<lock>	Allows the client to lock the entire configuration datastore system of a device	Lock the running configuration.
		`Request:`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<lock>`
		`<target>`
		`<running/>`
		`</target>`
		`</lock>`
		`</rpc>`
		
		`Response :`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		
		`<ok/>`
		`</rpc-reply>`

<Unlock>	Releases a previously locked configuration.
	An <unlock> operation will not succeed if either of the following conditions is true:
	• The specified lock is not currently active.
	• The session issuing the <unlock> operation is not the same session that obtained the lock.
		Lock and unlock the running configuration from the same session.
		`Request:`
		`rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<unlock>`
		`<target>`
		`<running/>`
		`</target>`
		`</unlock>`
		`</rpc>`
		
		`Response -`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

<close-session>	Closes the session. The server releases any locks and resources associated with the session and closes any associated connections.	Close a NETCONF session.
		`Request :`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<close-session/>`
		`</rpc>`
		
		`Response:`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

NETCONF Operation	Description	Example

<kill-session>	Terminates operations currently in process, releases locks and resources associated with the session, and close any associated connections.	Terminate a session if the ID is other session ID.
		`Request:`
		`<rpc message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<kill-session>`
		`<session-id>4</session-id>`
		`</kill-session>`
		`</rpc>`
		
		`Response:`
		`<rpc-reply message-id="101"`
		`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
		`<ok/>`
		`</rpc-reply>`

Note The system admin models support `<get>` and `<get-config>` operations, and only `<edit-config>` operations with the `<merge>` operation. The other operations such as `<delete>`, `<remove>`, and `<replace>` are not supported for the system admin models.

NETCONF Operation to Get Configuration

This example shows how a NETCONF <get-config> request works for CDP feature.

The client initiates a message to get the current configuration of CDP running on the router. The router responds with the current CDP configuration.

Netconf Request (Client to Router)	Netconf Response (Router to Client)

`<rpc message-id="101"`
`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
 	
`<get-config>`
`<source><running/></source>`
`<filter>`
`<cdp`
`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"/>`
`</filter>`
`</get-config>`
`</rpc>`
	`<?xml version="1.0"?>`
	`<rpc-reply message-id="101"`
	`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
	
	`<data>`
	`<cdp`
	`xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg">`
	
	`<timer>10</timer>`
	`<enable>true</enable>`
	`<log-adjacency></log-adjacency>`
	`<hold-time>200</hold-time>`
	`<advertise-v1-only></advertise-v1-only>`
	`</cdp>`
	`#22`
	`</data>`
	`</rpc-reply>`

The <rpc> element in the request and response messages enclose a NETCONF request sent between the client and the router. The `message-id` attribute in the <rpc> element is mandatory. This attribute is a string chosen by the sender and encodes an integer. The receiver of the <rpc> element does not decode or interpret this string but simply saves it to be used in the <rpc-reply> message. The sender

must ensure that the `message-id` value is normalized. When the client receives information from the server, the <rpc-reply> message contains the same `message-id`.

Retrieve Default Parameters Using with-defaults Capability

NETCONF servers report default data nodes in response to RPC requests in the following ways:

- report-all: All data nodes are reported
- trim: Data nodes set to the YANG default aren't reported
- explicit: Data nodes set to the YANG default by the client are reported

Cisco IOS XR routers support only the explicit basic mode. A server that uses this mode must consider any data node that isn't explicitly set to be the default data.

As per RFC 6243, the router supports `<with-defaults>` capability to retrieve the default parameters of configuration and state data node using a NETCONF protocol operation. The `<with-defaults>` capability indicates which default-handling basic mode is supported by the server. It also indicates support for additional retrieval modes. These retrieval modes allow a NETCONF client to control whether the server returns the default data.

By default, `<with-defaults>` capability is disabled. To enable this capability, use the following command in Config mode:

```
netconf-yang agent
 ssh
 with-defaults-support enable
!
```

Once enabled, the capability is applied to all netconf-yang requests.

After enabling, the router must return the new capability as:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults:1.0?basic-mode=explicit
```

The `<get>`, `<get-config>`, `<copy-config>` and ,`<edit-config>` operations support `with-defaults` capability.

Example 1: Create Operation

A valid `create` operation attribute for a data node that is set by the server to its schema default value must succeed. It is set or used by the device whenever the NETCONF client does not provide a specific value for the relevant data node. In the following example, an `edit-config` request is sent to create a configuration:

<edit-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:43efc290-c312-4df0-bb1b-a6e0bf8aac50">
<edit-config>
<target>
<candidate/>
</target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface>
<index>2</index>
<config>
```

```
<enabled xc:operation="create">false</enabled>
<index xc:operation="create">2</index>
</config>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</config>
</edit-config>
</rpc>
```

Response received from the NETCONF agent:

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

Commit the configuration.

```
[host 172.x.x.x session-id 2985924161] Requesting 'Commit'
[host 172.x.x.x session-id 2985924161] Sending:
<?xml version="1.0" encoding="UTF-8"?><nc:rpc
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:295eff87-1fb6-4f84-bb7d-c40b268eab1b"><nc:commit/></nc:rpc>

[host 172.x.x.x session-id 2985924161] Received:
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:295eff87-1fb6-4f84-bb7d-c40b268eab1b"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
CREATE operation completed
```

A `create` operation attribute for a data node that has been set by a client to its schema default value must fail with a `data-exists` error tag. The client can only create a default node that was not previously created by it. Else, the operation is rejected with the `data-exists` message.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1f29267f-7593-4a3c-8382-6ab9bec323ca">
<edit-config>
 <target>
  <candidate/>
 </target>
 <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
    <name>TenGigE0/0/0/0</name>
     <subinterfaces>
      <subinterface>
       <index>2</index>
        <config>
         <enabled xc:operation="create">false</enabled>
         <index xc:operation="create">2</index>
        </config>
      </subinterface>
     </subinterfaces>
    </interface>
   </interfaces>
 </config>
</edit-config>
</rpc>
```

```
[host 172.x.x.x session-id 2985924161] Received:
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:1f29267f-7593-4a3c-8382-6ab9bec323ca"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
   <error-tag>data-exists</error-tag>
   <error-severity>error</error-severity>
   <error-path
xmlns:ns1="http://openconfig.net/yang/interfaces">ns1:interfaces/ns1:interface[name =
'TenGigE0/0/0/0']/ns1:subinterfaces/ns1:subinterface[index = '2']/ns1:config</error-path>
 </rpc-error>
</rpc-reply>
```

Example 2: Delete Operation

A valid `delete` operation attribute for a data node set by a client to its schema default value must succeed. Whereas a valid `delete` operation attribute for a data node set by the server to its schema default value fails with a `data-missing` error tag.

<edit-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:de95a248-29d7-4030-8351-cef8b8d47cdb">
<edit-config>
<target>
<candidate/>
</target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface xc:operation="delete">
<index>2</index>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</config>
</edit-config>
</rpc>
```

Response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:de95a248-29d7-4030-8351-cef8b8d47cdb"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error>
<error-type>application</error-type>
<error-tag>data-missing</error-tag>
<error-severity>error</error-severity>
<error-path xmlns:ns1="http://openconfig.net/yang/interfaces">ns1:interfaces/ns1:
interface[name = 'TenGigE0/0/0/0']/ns1:subinterfaces/ns1:subinterface[index =
'2']/ns1:config</error-path></rpc-error>
</rpc-reply>
```

Example 3: Copy Configuration

In the following example, a `copy-config` request is sent to copy a configuration.

<copy-config> request sent to the NETCONF agent:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
 <target>
  <candidate/>
 </target>
 <source>
 <config>
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
    <name>TenGigE0/0/0/0</name>
    <subinterfaces>
     <subinterface>
      <index>2</index>
      <config>
       <index>2</index>
      </config>
     </subinterface>
    </subinterfaces>
   </interface>

  </interfaces>
</config>
</source>
<with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
</copy-config>
</rpc>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
  <commit/>
</rpc>
```

The show run command shows the copied configuration.

```
Router#show run
<data and time stamp>
Building configuration...
!! IOS XR Configuration 7.2.1
!! Last configuration change at <data and time stamp> by root
!
interface TenGigE0/0/0/0.2
!
end
```

Example 4: Get Configuration

The following example shows a `get-config` request with `explicit` mode to query the default parameters from the oc-interfaces.yang data model. The client gets the configuration values of what it sets.

<get-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:63a49626-9f90-4ebe-89fd-741410cddf29">
 <get-config>
 <source>
 <running/>
 </source>
 <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
 <filter type="subtree">
 <interfaces xmlns="http://openconfig.net/yang/interfaces"/>
 </filter>
```

```
 </get-config>
 </rpc>
```

<get-config> response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:99d8b2d0-ab05-474a-bc02-9242ba511308"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
   <interface>
     <name>TenGigE0/0/0/0</name>
       <subinterfaces>
       <subinterface>
         <index>2</index>
           <config>
            <index>2</index>
            <enabled>false</enabled>
           </config>
         <ipv6 xmlns="http://openconfig.net/yang/interfaces/ip">
         <config>
            <enabled>false</enabled>
          </config>
         </ipv6>
       </subinterface>
    </subinterfaces>
   </interface>
 <interface>
  <name>MgmtEth0/RSP0/CPU0/0</name>
   <config>
    <name>MgmtEth0/RSP0/CPU0/0</name>
     <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:ethernetCsmacd</type>

   </config>
    <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
     <config>
      <auto-negotiate>false</auto-negotiate>
     </config>
    </ethernet>
  <subinterfaces>
 <subinterface>
 <index>0</index>
<ipv4 xmlns="http://openconfig.net/yang/interfaces/ip">
<addresses>
 <address>
  <ip>172.xx.xx.xx</ip>
   <config>
    <ip>172.xx.xx.xx</ip>
    <prefix-length>24</prefix-length>
   </config>
 </address>
</addresses>
</ipv4>
</subinterface>
 </subinterfaces>
  </interface>
   <interface>
    <name>MgmtEth0/RSP1/CPU0/0</name>
    <config>
     <name>MgmtEth0/RSP1/CPU0/0</name>
     <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:ethernetCsmacd</type>
     <enabled>false</enabled>
  </config>
   <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
```

```
    <config>
      <auto-negotiate>false</auto-negotiate>
    </config>
  </ethernet>
 </interface>
 </interfaces>
 </data>
</rpc-reply>
READ operation completed
```

Example 5: Get Operation

The following example shows a `get` request with `explicit` mode to query the default parameters from the oc-interfaces.yang data model.

<get-config> request sent to the NETCONF agent:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:d8e52f0f-ceac-4193-89f6-d377ab8292d5">
<get>
<with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>
<filter type="subtree">
<interfaces xmlns="http://openconfig.net/yang/interfaces">
<interface>
<name>TenGigE0/0/0/0</name>
<subinterfaces>
<subinterface>
<index>2</index>
<state/>
</subinterface>
</subinterfaces>
</interface>
</interfaces>
</filter>
</get>
</rpc>
```

<get> response received from the NETCONF agent:

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:933df011-191f-4f31-9549-c4f7f6edd291"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
   <interfaces xmlns="http://openconfig.net/yang/interfaces">
    <interface>
    <name>TenGigE0/0/0/0</name>
     <subinterfaces>
      <subinterface>
      <index>2</index>
     <state>
      <index>2</index>
      <name>TenGigE0/0/0/0.2</name>
      <enabled>false</enabled>
      <admin-status>DOWN</admin-status>
      <oper-status>DOWN</oper-status>
      <last-change>0</last-change>
      <counters>
      <in-unicast-pkts>0</in-unicast-pkts>
      <in-pkts>0</in-pkts>
      <in-broadcast-pkts>0</in-broadcast-pkts>
      <in-multicast-pkts>0</in-multicast-pkts>
      <in-octets>0</in-octets>
```

```
      <out-unicast-pkts>0</out-unicast-pkts>
      <out-broadcast-pkts>0</out-broadcast-pkts>
      <out-multicast-pkts>0</out-multicast-pkts>
      <out-pkts>0</out-pkts>
      <out-octets>0</out-octets>
      <out-discards>0</out-discards>
      <in-discards>0</in-discards>
      <in-unknown-protos>0</in-unknown-protos>
      <in-errors>0</in-errors>
      <in-fcs-errors>0</in-fcs-errors>
      <out-errors>0</out-errors>
     <carrier-transitions>0</carrier-transitions>
    <last-clear>2020-03-02T15:35:30.927+00:00</last-clear>
   </counters>
  <ifindex>92</ifindex>
   <logical>true</logical>
  </state>
 </subinterface>
 </subinterfaces>
 </interface>
 </interfaces>
 </data>
</rpc-reply>
READ operation completed
```

Retrieve Transaction ID for NSO Operations

Table 2: Feature History Table

Feature Name	Release Information	Description
Unique Commit ID for Configuration State	Release 7.4.1	The network orchestrator is a central point of management for the network and typical workflow involves synchronizing the configuration states of the routers it manages. Loading configurations for comparing the states involves unnecessary data and subsequent comparisons are load intensive. This feature synchronizes the configuration states between the orchestrator and the router using a unique commit ID that the router maintains for each configuration commit. The orchestrator retrieves this commit ID from the router using NETCONF Remote Procedure Calls (RPCs) to identify whether the router has the latest configuration.

Cisco Network Services Orchestrator (NSO) is a data model-driven platform for automating your network orchestration. NSO uses NETCONF-based Network Element Drivers (NED) to synchronize the configuration

states of the routers it manages. NEDs comprise of the network-facing part of NSO and communicate over the native protocol supported by the router, such as Network Configuration Protocol (NETCONF).

IOS XR configuration manager maintains commit IDs (also known as the transaction IDs) for each commit operation. The manageability interfaces use these IDs. Currently, the operational data model provides a list of up to 100 last commits for NETCONF requests. The YANG client querying the last commit ID collects the entire list and finds the latest ID. Loading configurations for comparison to the orchestrator's configuration state can involve huge redundant data. The subsequent comparisons are also load intensive.

To overcome these limitations, the router maintains a unique last commit ID that is ideal for NSO operations. This ID indicates the latest configuration state on the router. The ID provides a one-step operation and increases the performance of configuration updates for the orchestrator.

An augmented configuration manageability model `Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper` provides a single `last-commit-id` for the unique commit state. This model is available as part of the base package.

The following table lists the synchronization support between NSO and the IOS XR variants:

Entity	64-bit Routers (Releases Earlier than 7.4.1)	64-bit Routers (Releases 7.4.1 and Later)
cfgmgr	Yes	Yes
sysadmin	Yes	Yes
cfgmgr-aug	No	Yes
Leaf Data	NA	cfgmgr-aug
Check synchronization (NSO functionality from release 7.4.1 and later)	No	Yes

Where:

- commit-id represents `Cisco-IOS-XR-config-cfgmgr-exec-oper:config-manager/global/config-commit/commits/commit/commit-id`
- cfgmgr is the XR configuration manager
- sysadmin represents the `Cisco-IOS-XR-sysadmin-system` data model
- cfgmgr-aug represents the `Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper` data model

The last commit ID is obtained from the configuration manager. The following example shows a sample NETCONF request and response to retrieve the commit ID:

```
Request:
<rpc message-id="test" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get>
  <filter type="subtree">
    <config-manager xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-oper">
      <global>
        <config-commit>
          <last-commit-id
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper"/>
        </config-commit>
```

```
      </global>
    </config-manager>
 </filter>
</get>
</rpc>


Response:
<rpc-reply message-id="test" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <config-manager xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-oper">
   <global>
    <config-commit>
     <last-commit-id
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-config-cfgmgr-exec-augmented-oper">
        XR:1000000009;Admin:1595-891537-949905</last-commit-id>
    </config-commit>
   </global>
  </config-manager>
 </data>
</rpc-reply>
```

Set Router Clock Using Data Model in a NETCONF Session

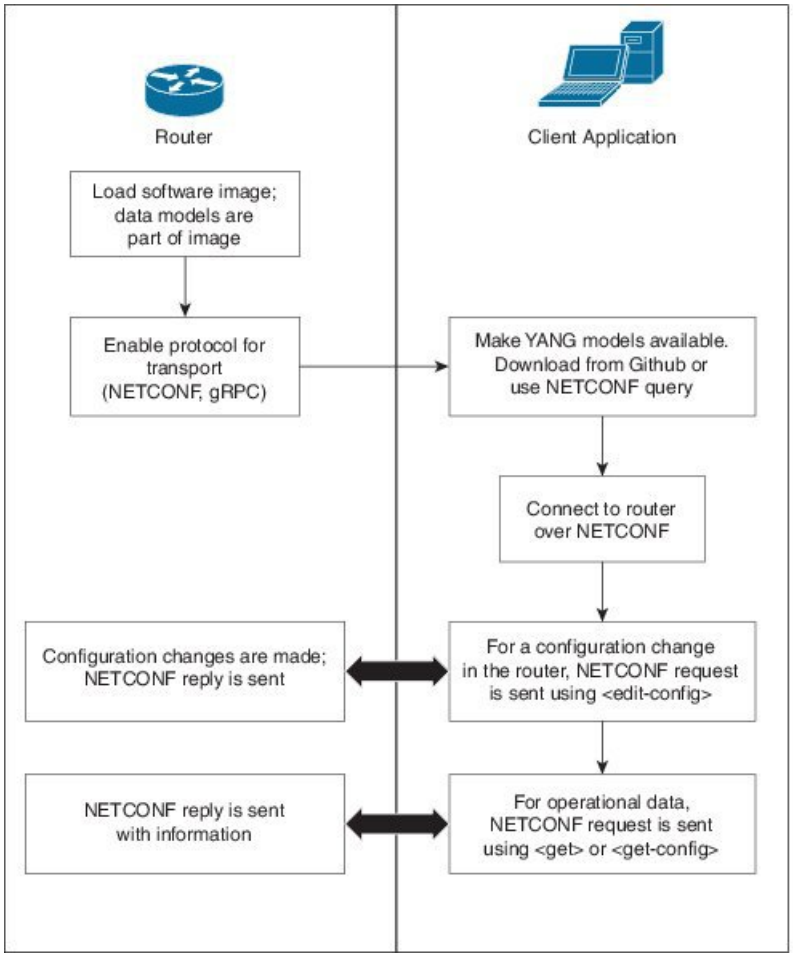
The process for using data models involves:

- Obtain the data models.
- Establish a connection between the router and the client using NETCONF communication protocol.
- Manage the configuration of the router from the client using data models.

Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization using **aaa authorization exec** command before setting up any configuration. For more information about configuring AAA authorization, see the *System Security Configuration Guide*.

The following image shows the tasks involved in using data models.

Figure 2: Process for Using Data Models

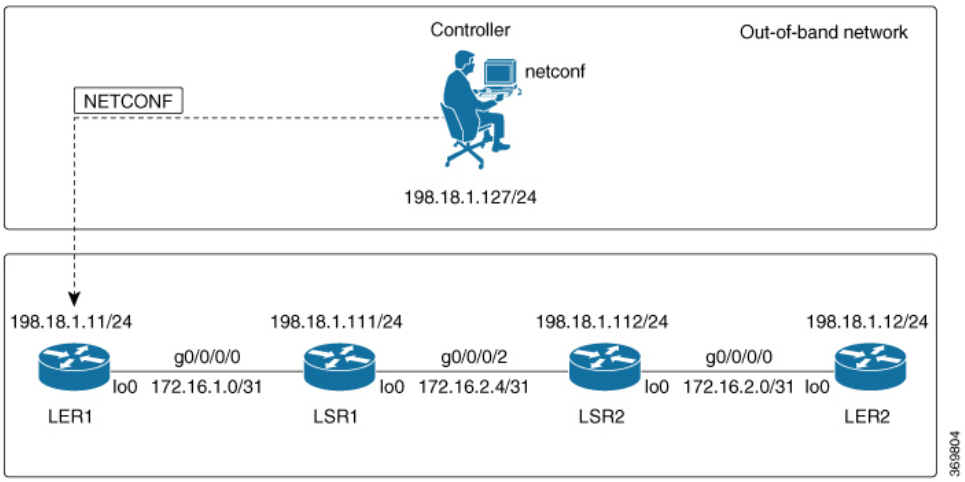


In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

Consider a network topology with four routers and one controller. The network consists of label edge routers (LER) and label switching routers (LSR). Two routers LER1 and LER2 are label edge routers, and two routers LSR1 and LSR2 are label switching routers. A host is the controller with a gRPC client. The controller communicates with all routers through an out-of-band network. All routers except LER1 are pre-configured with proper IP addressing and routing behavior. Interfaces between routers have a point-to-point configuration with `/31` addressing. Loopback prefixes use the format `172.16.255.x/32`.

The following image illustrates the network topology:

Figure 3: Network Topology for gRPC session



You use Cisco IOS XR native models `Cisco-IOS-XR-infra-clock-linux-cfg.yang` and `Cisco-IOX-XR-shellutil-oper` to programmatically configure the router clock. You can explore the structure of the data model using YANG validator tools such as pyang.

Before you begin

Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information

Procedure

Step 1 Explore the native configuration model for the system local time zone.

Example:

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-infra-infra-clock-linux-cfg.yang
module: Cisco-IOS-XR-infra-infra-clock-linux-cfg
    +--rw clock
        +--rw time-zone!
        +--rw time-zone-name string
        +--rw area-name string
```

Step 2 Explore the native operational state model for the system time.

Example:

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-shellutil-oper.yang
module: Cisco-IOS-XR-shellutil-oper
    +--ro system-time
        +--ro clock
        | +--ro year? uint16
        | +--ro month? uint8
        | +--ro day? uint8
        | +--ro hour? uint8
        | +--ro minute? uint8
        | +--ro second? uint8
        | +--ro millisecond? uint16
        | +--ro wday? uint16
```

```
        | +--ro time-zone? string
        | +--ro time-source? Time-source
        +--ro uptime
            +--ro host-name? string
            +--ro uptime? uint32
```

Step 3 Retrieve the current time on router LER1.

Example:

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ netconf get --filter xr-system-time-oper.xml
198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>17</hour>
        <minute>30</minute>
        <second>37</second>
        <millisecond>690</millisecond>
        <wday>1</wday>
        <time-zone>UTC</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>851237</uptime>
    </uptime>
</system-time>
```

Notice that the timezone `UTC` indicates that a local timezone is not set.

Step 4 Configure Pacific Standard Time (PST) as local time zone on LER1.

Example:

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
  <system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

Step 5 Verify that the router clock is set to PST time zone.

Example:

```
controller:netconf$ more xr-system-time-oper.xml
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>

controller:netconf$ netconf get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2018</year>
        <month>12</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

In summary, router LER1, which had no local timezone configuration, is programmatically configured using data models.