



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.

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
```

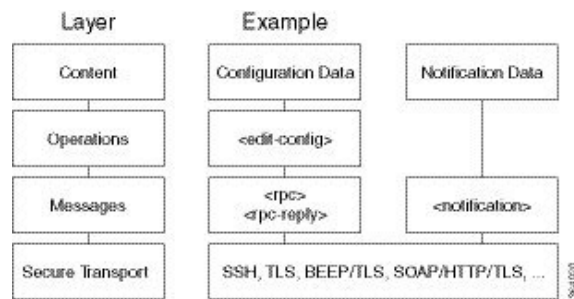
To enable the NETCONF SSH agent, use the following command:

```
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 3](#)
- [Set Router Clock Using Data Model in a NETCONF Session, on page 7](#)

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 <pre><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></pre>
<get>	Retrieves running configuration and device state information	Retrieve all acl configuration and device state information. <pre>Request: <get> <filter> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/> </filter> </get></pre>

NETCONF Operation	Description	Example
<edit-config>	Loads all or part of a specified configuration to the specified target configuration	<p>Configure ACL configs using Merge operation</p> <pre> <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> </pre>
<lock>	Allows the client to lock the entire configuration datastore system of a device	<p>Lock the running configuration.</p> <pre> 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> </pre>

NETCONF Operation	Description	Example
<Unlock>	<p>Releases a previously locked configuration.</p> <p>An <unlock> operation will not succeed if either of the following conditions is true:</p> <ul style="list-style-type: none"> • The specified lock is not currently active. • The session issuing the <unlock> operation is not the same session that obtained the lock. 	<p>Lock and unlock the running configuration from the same session.</p> <p>Request:</p> <pre>rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <unlock> <target> <running/> </target> </unlock> </rpc></pre> <p>Response -</p> <pre><rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply></pre>
<close-session>	Closes the session. The server releases any locks and resources associated with the session and closes any associated connections.	<p>Close a NETCONF session.</p> <p>Request :</p> <pre><rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <close-session/> </rpc></pre> <p>Response:</p> <pre><rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply></pre>
<kill-session>	Terminates operations currently in process, releases locks and resources associated with the session, and close any associated connections.	<p>Terminate a session if the ID is other session ID.</p> <p>Request:</p> <pre><rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <kill-session> <session-id>4</session-id> </kill-session> </rpc></pre> <p>Response:</p> <pre><rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply></pre>



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)
<pre><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></pre>	<pre><?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-vl-only></advertise-vl-only> </cdp> #22 </data> </rpc-reply></pre>

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`.



Note

- From 7.0.x, **cg** `<var>` configurations are not supported under interfaces.
- The command **hw-module service sesh** is not supported.

Set Router Clock Using Data Model in a NETCONF Session

The process for using data models involves:

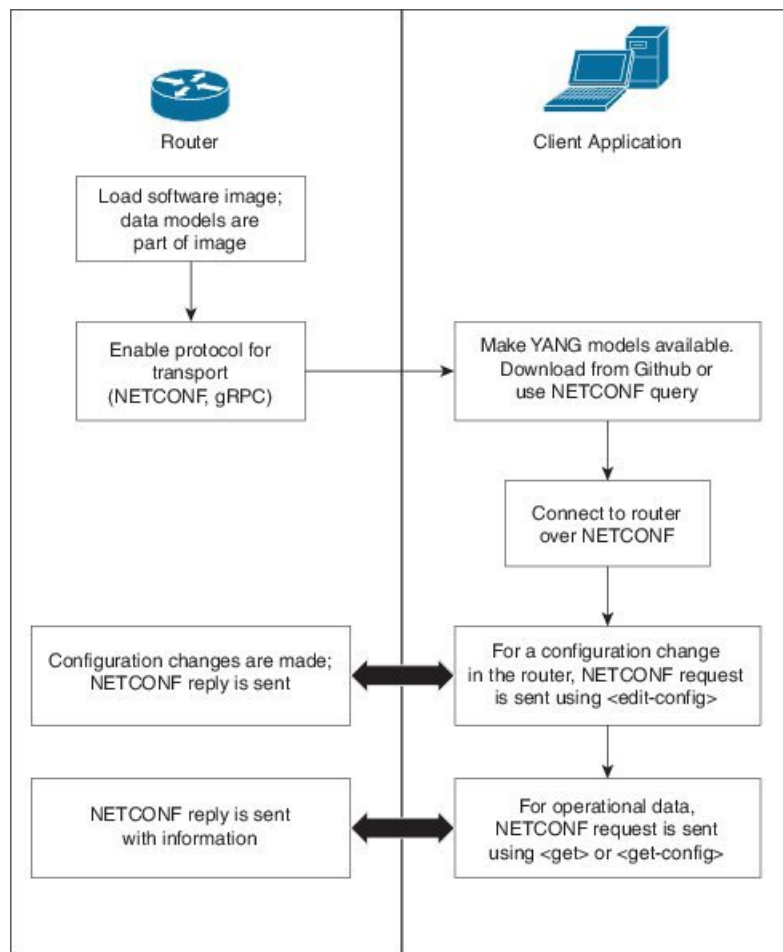
- Obtain the data models.
- Establish a connection between the router and the client using NETCONF communication protocol.
- Manage the configuration of the router from the client using data models.



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



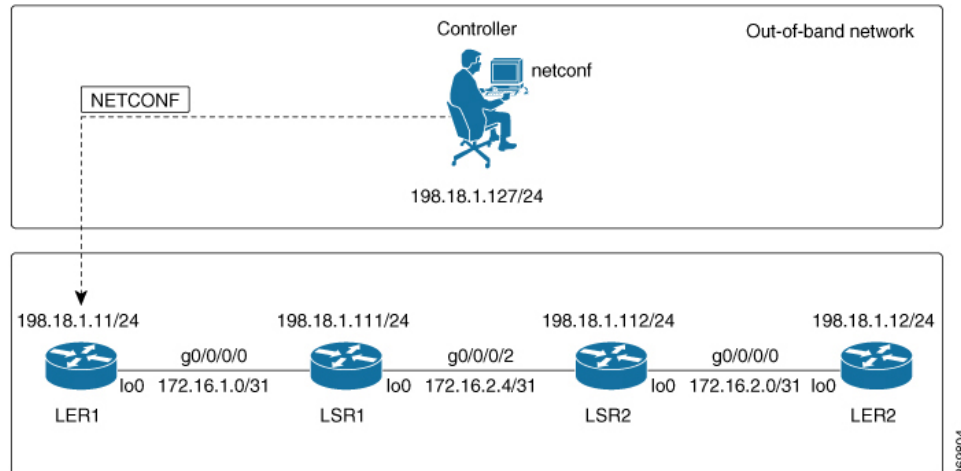
In this section, you use native data models to configure the router clock and verify the clock state using a NETCONF session.

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

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>

```

```
</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.
