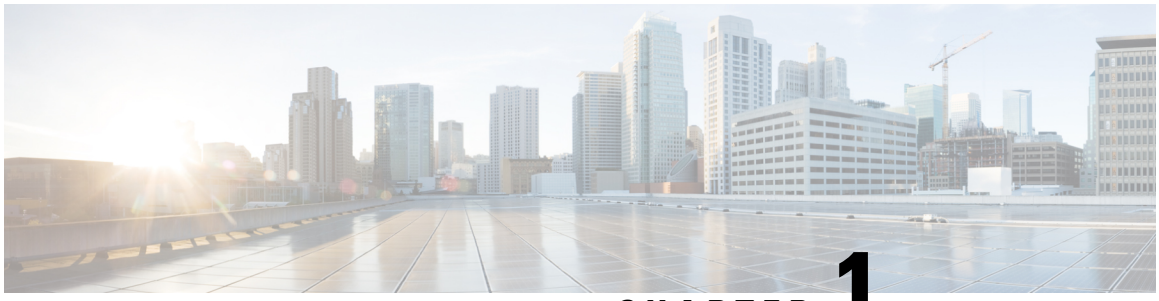
Cisco NCS 2000 Series SVO Data Models Configuration Guide, Release 12.0.1

First Published: 2020-07-31

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883



CHAPTER 1

Data Models—Scope, Need, and Benefits

Scope

Data models can be used to automate configuration tasks across heterogeneous devices in a network.

Data models handle the following types of requirements on routers:

- **Configuration data:** A set of writable data that is required to transform a system from an initial default state into its current state. For example, configuring entries of the IP routing tables, configuring the interface MTU to use a specific value, configuring an ethernet interface to run at a given speed, and so on.
- **Operational state data:** A set of data that is obtained by the system at runtime and influences the behavior of the system in a manner similar to configuration data. However, in contrast to configuration data, operational state data is transient. The data is modified by interactions with internal components or other systems using specialized protocols. For example, entries obtained from routing protocols such as OSPF, attributes of the network interfaces, and so on.
- **Actions:** A set of NETCONF actions that support robust network-wide configuration transactions. When a change is attempted that affects multiple devices, the NETCONF actions simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.

Data models provide a well-defined hierarchy of the configurational and operational data of a router, and NETCONF actions. The data models are programmed to provide a common framework of configurations to be deployed across networks. This common framework helps to program and manage a network with ease.

Need

Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly.

CLIs are widely used for configuring and extracting the operational details of a router. But the general mechanism of CLI scraping is not flexible and optimal. Small changes in the configuration require rewriting scripts multiple times. Bulk configuration changes through CLIs are cumbersome and error-prone. These limitations restrict automation and scale.

To overcome these limitations, Cisco IOS XR supports a programmatic way of writing configurations to any network device using data models.

Data models help to manipulate configuration data, retrieve operational data, and perform actions. The data models replace the process of manual configuration and are written in an industry-defined language. Although

configurations using CLIs are easier and human-readable, automating the configuration using data models results in scalability.

The data models provides access to the capabilities of the devices in a network using Network Configuration Protocol (NETCONF) protocol. The operations on the router are carried out by the protocols using YANG models to automate and programme operations in a network.

The process of automating configurations in a network is accomplished using the core components - router, client application, YANG model and communication protocols.

Benefits

Configuring routers using data models overcomes drawbacks posed by traditional router management because the data models:

- Provide a common model for configuration and operational state data, and perform NETCONF actions.
- Use protocols to communicate with the routers to get, manipulate and delete configurations in a network.
- Automate configuration and operation of multiple routers across the network.

- Process for using Data Models, on page 2

Process for using Data Models

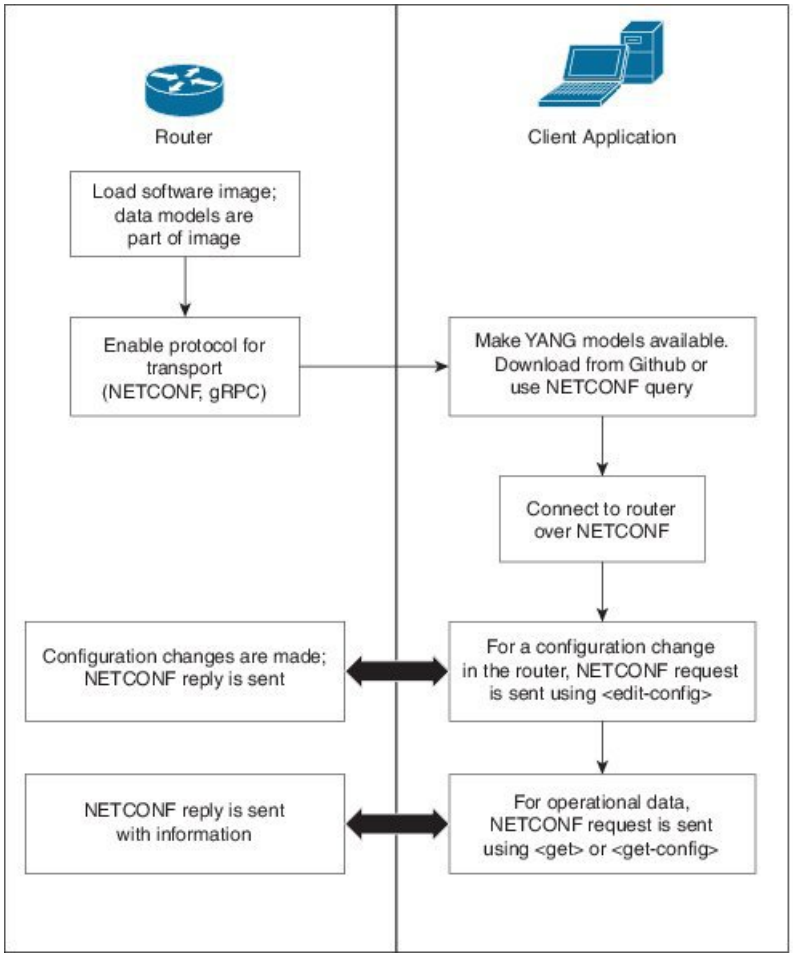
The process for using data models involves:

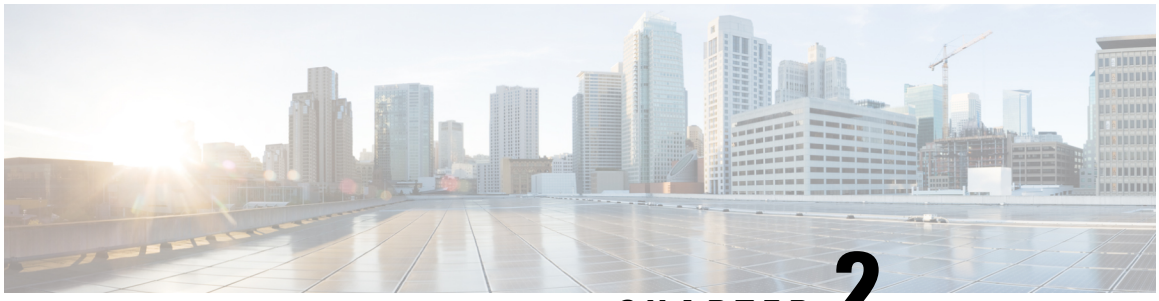
- Obtain the data models.
- Establish a connection between the router and the client using communication protocols such as NETCONF.
- Manage the configuration of the router from the client using data models.

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.

Figure 1 shows the tasks involved in using data models.

Figure 1: Process for Using Data Models





CHAPTER 2

YANG Models

A YANG model defines a data model through the data of the router, and the hierarchical organization and constraints on that data. Each module is uniquely identified by a namespace URL. The YANG models describe the configuration and operational data, perform actions, remote procedure calls, and notifications for network devices.

The YANG models must be obtained from the router. The models define a valid structure for the data that is exchanged between the router and the client. The models are used by NETCONF and gRPC-enabled applications.

YANG models can be:

- **Cisco-specific models:** For a list of supported models and their representation, see https://github.com/YangModels/yang/tree/master/vendor/cisco/svo.
- **Common models:** These models are industry-wide standard YANG models from standard bodies, such as IETF and IEEE. These models are also called Open Config (OC) models. Like synthesized models, the OC models have separate YANG models defined for configuration data and operational data, and actions.

For a list of supported OC models and their representation, see https://github.com/YangModels/yang/tree/master/vendor/cisco/svo.

- Components of a YANG Model, on page 5
- Structure of YANG Models, on page 6
- Communication Protocols, on page 6

Components of a YANG Model

A YANG model defines a single data model. However, a module can reference definitions in other modules and sub-modules by using one of these statements:

- **import** imports external modules
- **include** includes one or more sub-modules
- **augment** provides augmentations to another module, and defines the placement of new nodes in the data model hierarchy
- **when** defines conditions under which new nodes are valid

- **prefix** references definitions in an imported module

The YANG models configure a feature, retrieve the operational state of the router, and perform actions.

Structure of YANG Models

YANG data models can be represented in a hierarchical, tree-based structure with nodes. This representation makes the models easy to understand.

Each feature has a defined YANG model, which is synthesized from schemas. A model in a tree format includes:

- Top level nodes and their subtrees
- Subtrees that augment nodes in other YANG models
- Custom RPCs

YANG defines four node types. Each node has a name. Depending on the node type, the node either defines a value or contains a set of child nodes. The nodes types for data modeling are:

- leaf node - contains a single value of a specific type
- leaf-list node - contains a sequence of leaf nodes
- list node - contains a sequence of leaf-list entries, each of which is uniquely identified by one or more key leaves
- container node - contains a grouping of related nodes that have only child nodes, which can be any of the four node types

Communication Protocols

Communication protocols establish connections between the router and the client. The protocols help the client to consume the YANG data models to, in turn, automate and programme network operations.

YANG uses the Network Configuration Protocol (NETCONF) protocol.

The transport and encoding mechanisms for this protocol is shown in the table:

Protocol	Transport	Encoding/ Decoding
NETCONF	ssh	xml

NETCONF Protocol

NETCONF provides mechanisms to install, manipulate, or delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. Use **ssh server capability netconf-xml** command to enable NETCONF to reach XML subsystem via port 22. NETCONF 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.

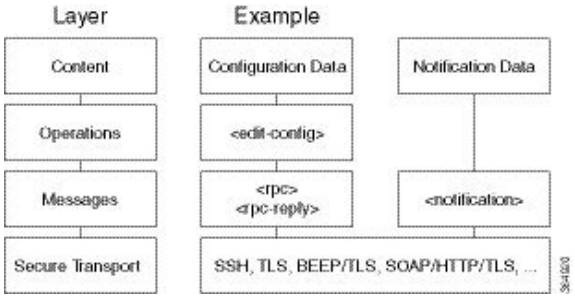
NETCONF Session

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. Sessions are terminated using *close* or *kill* messages.

NETCONF Layers

NETCONF can be partitioned into four layers:

Figure 2: 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

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
```

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>
<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>

NETCONF Operation	Description	Example
<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>`

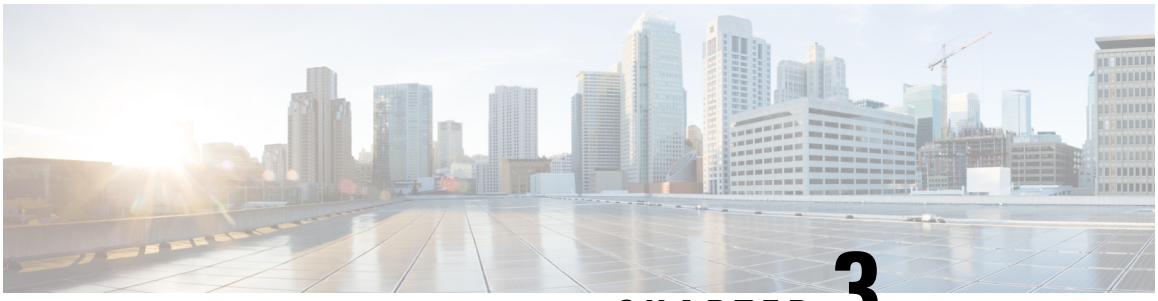
<lock>	Allows the client to lock the entire configuration datastore system of a device	Lock the running configuration.
		`Request:`
		`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <lock> <target> <running/> </target> </lock> </rpc>`
		
		`Response :`
		`<rpc-reply message-id="101"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

NETCONF Operation	Description	Example

<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>`

<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>`



CHAPTER 3

Obtain Data Models

The data models are available in the mgbl pie software package. Installing a package on the router installs specific features that are part of that package. Cisco IOS XR software is divided into various software packages to select the features to run on the router. Each package contains components that perform a specific set of router functions, such as routing, security, and so on.

1. Verify that the data models available are using `netconf-monitoring` request.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <get>
   <filter type="subtree">
     <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
     <schemas/>
     </netconf-state>
   </filter>
 </get>
</rpc>
```

All IOS XR and System Admin YANG models are displayed.

The models are in the `.yang` format. A model with:

- `-oper` in the model name indicates an operational model.
- `-cfg` indicates a configuration model.
- `-act` indicates a NETCONF actions model. For example, Cisco-IOS-XR-ipv4-ospf-act.yang is an action model for OSPF.

  The YANG models can be retrieved from the router without logging into the router using **get-schema** command:

Get Schema List (data will be used in step 2).

```
<get>
<filter type="subtree">
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas/>
</netconf-state>
</filter>
</get>
</rpc>
```

```
TRACE: 2016/06/13 11:11:42 transport.go:104: Reading from connection
TRACE: 2016/06/13 11:11:42 gnc_main.go:587: Session established (Id: 1009461378)
TRACE: 2016/06/13 11:11:42 session.go:93: Request:
<rpc message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
```

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get><filter type="subtree"><netconf-state

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring"><schemas/></netconf-state></filter></get></rpc>
TRACE: 2016/06/13 11:11:42 transport.go:104: Reading from connection
TRACE: 2016/06/13 11:11:42 session.go:117:
Response:
#143589
<rpc-reply message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas>
<schema>
<identifier>Cisco-IOS-XR-crypto-sam-oper</identifier>
<version>2015-01-07</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
<location>NETCONF</location>
</schema>
<schema>
<identifier>Cisco-IOS-XR-crypto-sam-oper-sub1</identifier>
<version>2015-01-07</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
<location>NETCONF</location>
</schema>
<schema>
<identifier>Cisco-IOS-XR-snmp-agent-oper</identifier>
<version>2015-10-08</version>
<format>yang</format>
<namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
<location>NETCONF</location>
</schema>
------------<truncated>--------------
```

All the models on the router are displayed.

Note You can register at https://wwwin-github.cisco.com and then refer the following link for SVO Yang models:https://github.com/YangModels/yang/tree/master/vendor/cisco/svo

• Commit Configuration, on page 12

Commit Configuration

Commit the configuration to set the new values in the current running configuration.

The configuration can also be committed through a `confimed-commit` operation. NETCONF supports confirmed-commit RPC. This RPC requires an explicit confirmation from the user before the configuration takes effect on the router. This feature helps in verifying that the change in the configuration works correctly and does not cause fluctuation in the management connectivity. If the configuration change causes loss of management connectivity, the configuration is automatically rolled back to the previous committed configuration after the default `confirm-timeout` period of 600 seconds.

To commit a configuration, use </commit> RPC:

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

```
    <commit/>
</rpc>
```

To confirm-commit a configuration:

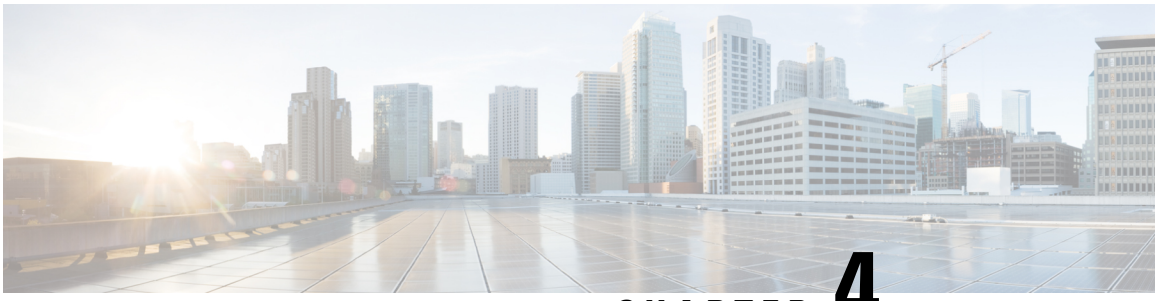
```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
    </commit>
</rpc>
```

The confirmed-commit capability supports the <cancel-commit> operation and the <confirmed>, <confirm-timeout>, <persist>, and <persist-id> parameters for the <commit> operation.

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <commit>
         <confirmed/>
         <persist>IQ,d4668</persist>
         <confirm-timeout>120</confirm-timeout>
       </commit>
</rpc>
```

A confirmed-commit request will fail with `Datastore Locked` error if:

- Another operation is performed between a confirmed-commit and a confirming-commit operation
- Another session has an active confirmed-commit request and a persist ID was not provided
- A persist ID was provided but did not match the persist ID of the active confirmed-commit session



CHAPTER 4

Examples of SVO Data Models

The examples of SVO data models are as follows:

Example 1: The following example shows the RPC Request and RPC Response messages to get information about a particular node.

RPC Request

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="7">
    <get>
        <filter>
            <svo xmlns="http://cisco.com/yang/svo">
                <node-information></node-information>
            </svo>
        </filter>
    </get>
</rpc>
```

RPC Response

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="7">
    <data>
        <svo xmlns="http://cisco.com/yang/svo">
            <node-information>
                <name>SANITY_DEVICE_263</name>
                <optical-type
xmlns:ccet="http://cisco.com/yang/svo/common-equipment-types">ccet:roadm</optical-type>
                <network-config>
                    <ip-address>10.58.226.118</ip-address>
                </network-config>
                <tdm-terminology-type
xmlns:ccet="http://cisco.com/yang/svo/common-equipment-types">ccet:ansi</tdm-terminology-type>

                <sw-version>12.1.0.B0263</sw-version>
                <admin-plane-sw-version>12.1.0.B0263</admin-plane-sw-version>
                <ha-manager-sw-version>12.1.0.B0263</ha-manager-sw-version>
                <time-settings>
                    <enable-date-and-time>false</enable-date-and-time>
                    <ntp-svo>
                        <server-address>10.58.228.2</server-address>
                        <backup-server-address>10.58.228.3</backup-server-address>
                    </ntp-svo>
                    <ntp-devices>
                        <primary-server>0.0.0.0</primary-server>
                        <secondary-server>0.0.0.0</secondary-server>
                    </ntp-devices>
```

```
                    <ntp-card-controllers>
                        <server-address>0.0.0.0</server-address>
                        <backup-server-address>0.0.0.0</backup-server-address>
                    </ntp-card-controllers>
                    <time-zone>UTC</time-zone>
                </time-settings>
            </node-information>
        </svo>
    </data>
</rpc-reply>
```

Example 2: The following example shows the RPC Request and RPC Response messages to lock the running datastore.

RPC Request

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="567">
<lock>
<target>
<running></running>
</target>
</lock>
</rpc>
```

RPC Response

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="567"><ok/></rpc-reply>
```