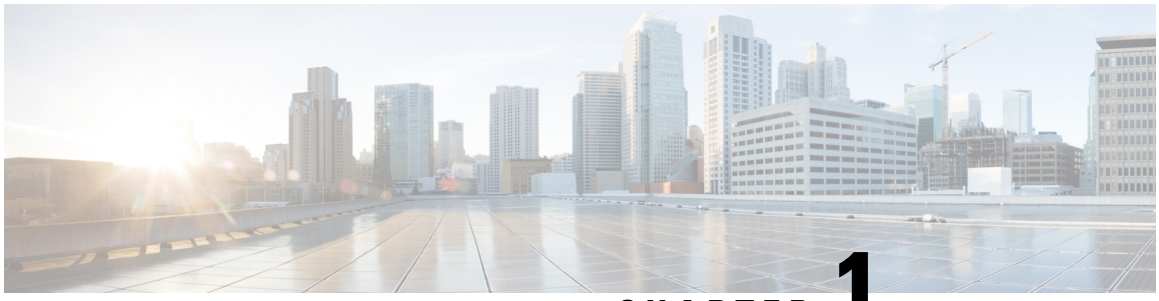
Data Models Configuration Guide for Cisco NCS 1001, Cisco IOS XR Releases

First Published: 2017-07-14

Last Modified: 2023-03-30

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

CONTENTS



CHAPTER 1

Data models

Data modeling is standard based approach to model configuration and operational data in networking devices. Using data models, customers can automate and simplify network wide visibility and configuration.

Data Models - Programmatic and Standards-based Configuration

Cisco IOS XR software supports the automation of configuration of multiple routers across the network using Data models. Configuring routers using data models overcomes drawbacks posed by traditional router management techniques.

CLIs are widely used for configuring a router and for obtaining router statistics. Other actions on the router, such as, switch-over, reload, process restart are also CLI-based. Although, CLIs are heavily used, they have many restrictions.

Customer needs are fast evolving. Typically, a network center is a heterogenous mix of various devices at multiple layers of the network. Bulk and automatic configurations need to be accomplished. CLI scraping is not flexible and optimal. Re-writing scripts many times, even for small configuration changes is cumbersome. Bulk configuration changes through CLIs are error-prone and may cause system issues. The solution lies in using data models - a programmatic and standards-based way of writing configurations to any network device, replacing the process of manual configuration. Data models are written in a standard, industry-defined language. Although configurations using CLIs are easier (more human-friendly), automating the configuration using data models results in scalability.

Cisco IOS XR supports the YANG data modeling language. YANG can be used with Network Configuration Protocol (NETCONF) to provide the desired solution of automated and programmable network operations.

YANG model

YANG is a data modeling language used to describe configuration and operational data, remote procedure calls and notifications for network devices. The salient features of YANG are:

- Human-readable format, easy to learn and represent

- Supports definition of operations
- Reusable types and groupings
- Data modularity through modules and submodules
- Supports the definition of operations (RPCs)
- Well-defined versioning rules
- Extensibility through augmentation

For more details of YANG, refer RFC 6020 and 6087.

NETCONF and gRPC (Google Remote Procedute Call) provide a mechanism to exchange configuration and operational data between a client application and a router and the YANG models define a valid structure for the data (that is being exchanged).

Protocol	Transport	Encoding/ Decoding
NETCONF	SSH	XML
gRPC	HTTP/2	XML, JSON

Each feature has a defined YANG model. Cisco-specific YANG models are referred to as synthesized models. Some of the standard bodies, such as IETF , IEEE and Open Config, are working on providing an industry-wide standard YANG models that are referred to as common models.

Components of a YANG Model

A module defines a single data model. However, a module can reference definitions in other modules and submodules by using the **import** statement to import external modules or the **include** statement to include one or more submodules. A module can provide augmentations to another module by using the **augment** statement to define the placement of the new nodes in the data model hierarchy and the **when** statement to define the conditions under which the new nodes are valid. **Prefix** is used when referencing definitions in the imported module.

YANG models are available for configuring a feature and to get operational state (similar to show commands)

This is the configuration YANG model for AAA (denoted by - cfg)

```
(snippet)
module Cisco-IOS-XR-aaa-locald-cfg {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-cfg";


  prefix "aaa-locald-cfg";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  import Cisco-IOS-XR-aaa-lib-cfg { prefix "a1"; }

  /*** META INFORMATION ***/
```

```
  organization "Cisco Systems, Inc.";
          ......................
          ...................... (truncated)
```

This is the operational YANG model for AAA (denoted by -oper)

```
 (snippet)
module Cisco-IOS-XR-aaa-locald-oper {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-oper";


  prefix "aaa-locald-oper";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  include Cisco-IOS-XR-aaa-locald-oper-sub1 {
    revision-date 2015-01-07;
  }

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";
    ......................
    ...................... (truncated)
```

Note A module may include any number of sub-modules, but each sub-module may belong to only one module. The names of all standard modules and sub-modules must be unique.

Structure of YANG models

YANG data models can be represented in a hierarchical, tree-based structure with nodes, which makes them more easily understandable. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes. The nodes types (for data modeling) are:

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

Data Types

YANG defines data types for leaf values. These data types help the user in understanding the relevant input for a leaf.

Name	Description
binary	Any binary data
bits	A set of bits or flags
boolean	"true" or "false"
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	Enumerated strings
identityref	A reference to an abstract identity
instance-identifier	References a data tree node
int (integer-defined values)	8-bit, 16-bit, 32-bit, 64-bit signed integers
leafref	A reference to a leaf instance
uint	8-bit, 16-bit, 32-bit, 64-bit unsigned intergers
string	Human-readable string
union	Choice of member types

Supported YANG models

The complete list of the supported IOSXR YANG models are:

`https://github.com/YangModels/yang/tree/master/vendor/cisco/xr`

Introduction to NETCONF

NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. 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 typically running as part of a network manager. The server is typically a network device (router).

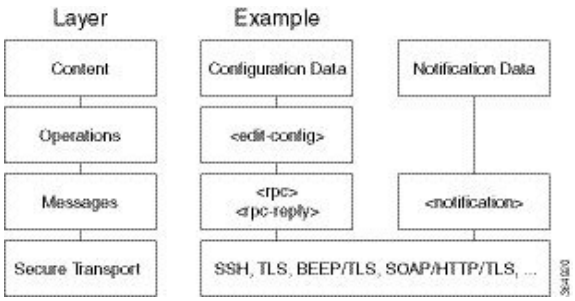
For more NETCONF details, refer RFC 6241.

NETCONF sessions

A NETCONF session is the logical connection between a network administrator or network configuration application and a network device. Global configuration attributes can be changed during any authorized session, and 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 the close or kill messages.

NETCONF Layers

Figure 1: NETCONF Layers



NETCONF can be partitioned into four 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 server

NETCONF Operations

NETCONF defines the existence of one or more configuration datastores and allows configuration operations on them. A configuration datastore is defined as the 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.

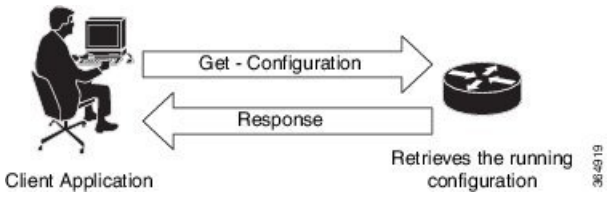
IOS XR NETCONF supports the following operations:

- <get-config>—Retrieves all or part of a specified configuration from a named data store
- <get>—Retrieves running configuration and device state information
- <edit-config>—Loads all or part of a specified configuration to the specified target configuration
- <get-schema>—Retrieves the required schema from the router

NETCONF Operations: Example

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

Figure 2: <get-config> request



The send message request is to get the current configuration of CDP running on the router. The return message includes the current CDP configuration.

NETCONF request (client to server)	NETCONF reply (server 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> </data> </rpc-reply>

The RPC element is used to enclose a NETCONF request sent from the client to the server. The <rpc> element has a mandatory attribute *message-id*, which is a string chosen by the sender of the RPC that will commonly encode a monotonically increasing integer. The receiver of the RPC does not decode or interpret this string but simply saves it to be used as a *message-id* attribute in any resulting <rpc-reply> message. The sender MUST ensure that the *message-id* value is normalized. The RPC reply message contains the same message-id when the client receives information from the server.

Subtree Filtering

XML subtree filtering is a mechanism that allows an application to select particular XML subtrees to include in the <rpc-reply> for a <get> or <get-config> operation.

Subtree Filter Components

A subtree filter is comprised of XML elements and their XML attributes. Elements that can be present in a subtree filter are:

- Namespace selection - A namespace is considered to match (for filter purposes) if the XML namespace associated with a particular node within the filter element is the same as in the underlying data model. A namespace selection cannot be used by itself; at least one element must be specified in the filter if any elements are to be included in the filter output.

  ```
  Example:

  <filter type="subtree">
     <top xmlns="http://example.com/schema/1.2/config"/>
  </filter>
  ```

- Attribute match expressions -Filtering is done by matching a specified attribute value. This filtering with the *Match* attribute can be specified only in Table classes.

  ```
   Example:
  ifName is the attribute match expression
  <filter type="subtree">
         <t:top xmlns:t="http://example.com/schema/1.2/config">
           <t:interfaces>
             <t:interface t:ifName="eth0"/>
           </t:interfaces>
         </t:top>
       </filter>
  ```

- Containment Nodes - Filtering is done by specifying nodes (classes) that have child nodes (classes). This filtering is by specifying container classes.

```
 Example: top is a containment node
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users/>
       </top>
     </filter>
```

- Selection Nodes - Filtering is done by specifying leaf nodes. This filtering specifies leaf classes.

```
 Example: users is a selection node (in the containment node - top)
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users/>
       </top>
     </filter>
```

- Content Match Nodes - Filtering is done by exactly matching the content of a leaf node. This filtering is done by specifying naming the class value for table classes.

```
Example: name is the content match node (in the containment node - top and the selection
 node - user)
<filter type="subtree">
       <top xmlns="http://example.com/schema/1.2/config">
         <users>
           <user>
             <name>fred</name>
           </user>
         </users>
       </top>
     </filter>
```

gRPC

gRPC is a language-neutral, open source, RPC (Remote Procedute Call) system developed by Google. By default, it uses protocol buffers as the binary serialization protocol. It can be used with other serialization protocols as well such as JSON, XML etc. The user needs to define the structure by defining protocol buffer message types in*.proto* files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs.

gRPC encodes requests and responses in binary. Although Protobufs was the only format supported in the initial release, gRPC is extensible to other content types. The Protobuf binary data object in gRPC is transported using HTTP/2 (RFC 7540). HTTP/2 is a replacement for HTTP that has been optimized for high performance. HTTP/2 provides many powerful capabilities including bidirectional streaming, flow control, header compression and multi-plexing. gRPC builds on those features, adding libraries for application-layer flow-control, load-balancing and call-cancellation.

gRPC supports distributed applications and services between a client and server. gRPC provides the infrastructure to build a device management service to exchange configuration and operational data between a client and a server in which the structure of the data is defined by YANG models.

Cisco gRPC IDL

The protocol buffers interface definition language (IDL) is used to define service methods, and define parameters and return types as protocol buffer message types.

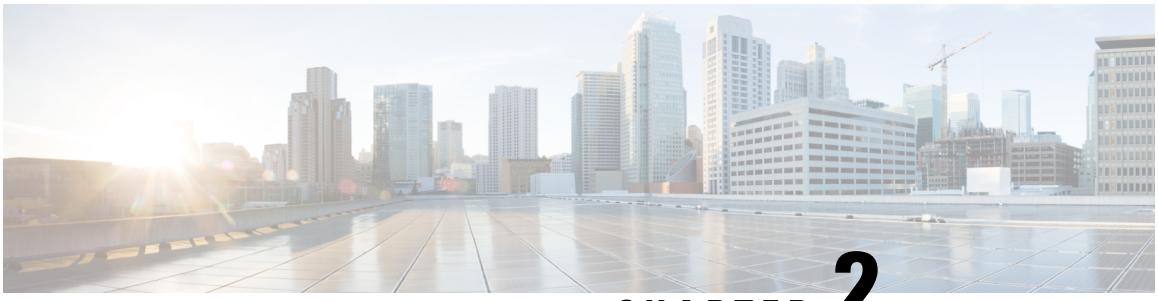
gRPC requests can be encoded and sent across to the router using JSON. gRPC IDL also supports the exchange of CLI.

For gRPC transport, gRPC IDL is defined in .proto format. Clients can invoke the RPC calls defined in the IDL to program XR. The supported operations are - Get, Merge, Delete, Replace. The gRPC JSON arguments are defined in the IDL.

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

    rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};

    rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};

    rpc DeleteConfig(ConfigArgs) returns(ConfigReply) {};

    rpc ReplaceConfig(ConfigArgs) returns(ConfigReply) {};

    rpc CliConfig(CliConfigArgs) returns(CliConfigReply) {};

}
```

gRPC Operations

- oper get-config—Retrieves a configuration
- oper merge-config— Appends to an existing configuration
- oper delete-config—Deletes a configuration
- oper replace-config—Modifies a part of an existing configuration
- oper get-oper—Gets operational data using JSON
- oper cli-config—Performs a configuration
- oper showcmdtextoutput



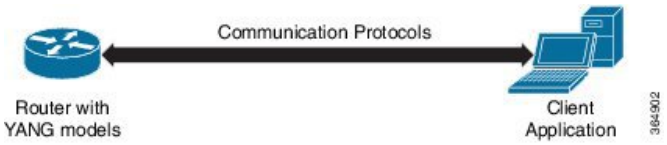
CHAPTER 2

Using Data models

This section explains the required configurations and procedures for using data models.

Using Data models

Figure 3: Workflow for using Data models



The above illustration gives a quick snap shot of how YANG can be used with Netconf in configuring a network device using a client application.

The tasks that help the user to implement Data model configuration are listed here.

1. Load the software image ; the YANG models are a part of the software image. Alternatively, the YANG models can also be downloaded from:

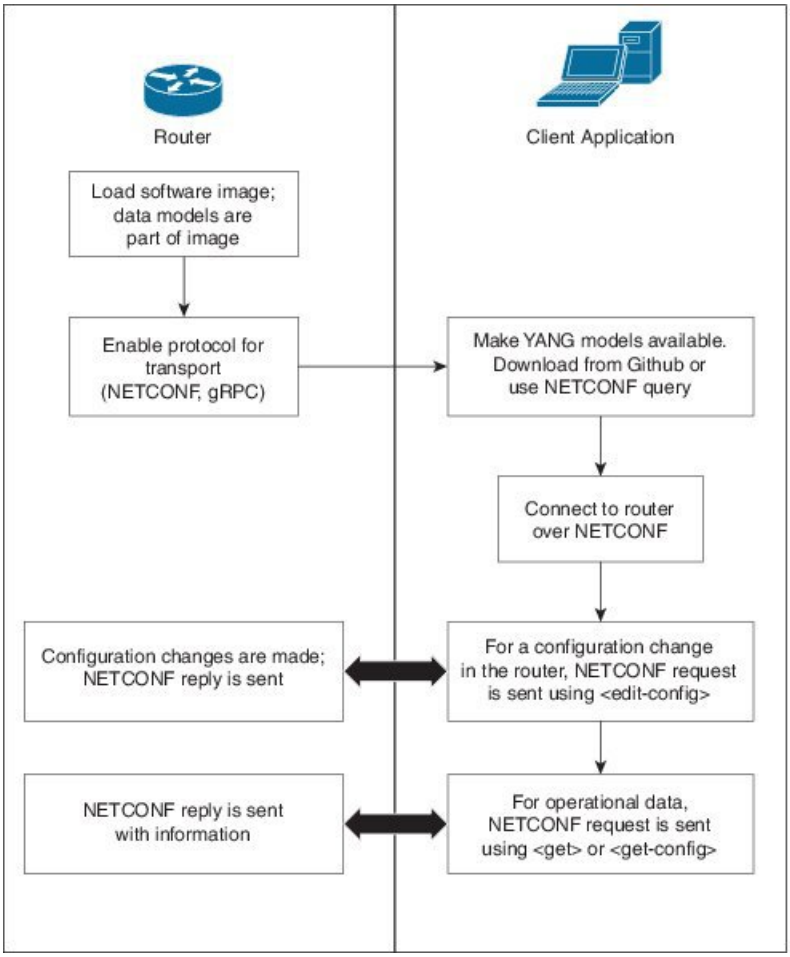
   ```
   https://github.com/YangModels/yang/tree/master/vendor/cisco/xr
   ```

   Users can also query using NETCONF to get the list of models.

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <rpc message-id="100" 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>
   ```

2. Communication between the router and the application happens by SSH on Netconf. Enable Netconf on the router on a suitable port.
3. From the client application, connect to the router using Netconf on SSH. Run Netconf operations to make configuration changes or get operational data.

Figure 4: Lane Diagram to show the router and client application operations



Enabling Netconf

This task enables Netconf over SSH.

Before you begin

- Install the required packages (k9sec and mgbl)
- Generate relevant crypto keys

Step 1 **netconf-yang agent ssh**

Enables the Netconf agent process.

Step 2 **ssh server netconf**

Enables Netconf.

Step 3 **ssh server v2**

Enables SSH on the device and enables Netconf on port 22 if the Netconf agent process is enabled.

What to do next

The **netconf-yang agent session** command enables the user to set session parameters.

```
netconf-yang agent session {limit value |  absolute-timeout value |  idle-timeout value}
```

where,

- **limit** *value*- sets the maximum count for concurrent netconf-yang sessions. Range is 1 to 1024. The default value is 50.
- **absolute-timeout** *value*- sets the absolute session lifetime. Range is 1 to 1440 (in minutes).
- **idle-timeout** *value*- sets the idle session lifetime. Range is 1 to 1440 (in minutes).

Enabling gRPC

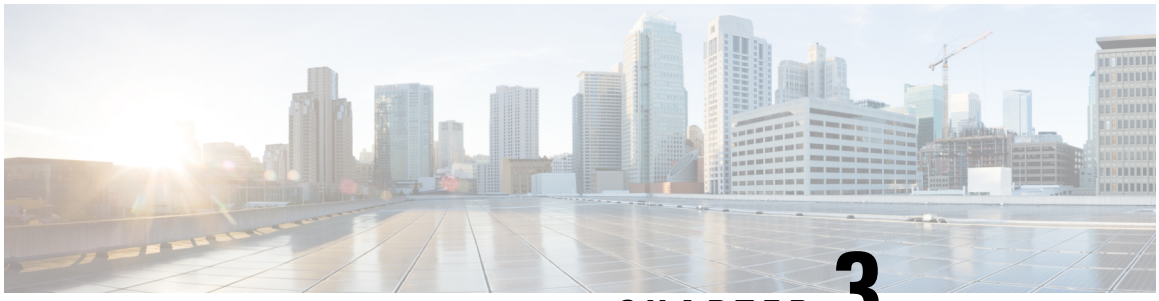
Use the following procedure to enable gRPC over HTTPS/2. gRPC supports both, the IPv4 and IPv6 address families (default is IPv4).

Step 1 Install the GO client. For more details on installing the GO client, see https://golang.org/doc/install.

Step 2 Configure the gRPC port, using the **grpc port** command.

```
RP/0/RP0/CPU0:ios(config)#grpc
RP/0/RP0/CPU0:ios(config)#port 57400
RP/0/RP0/CPU0:ios(config)#tls
RP/0/RP0/CPU0:ios(config)#commit
```

Port can range from 57344 to 57999. If a port is unavailable, an error is displayed.



CHAPTER 3

Configuring NCS 1001 Using Data Models

This section includes examples for configuring NCS 1001 using Data models.

Supported YANG Models in NCS 1001

Table 1: Feature History

Feature Name	Release	Description
New Optics Telemetry Bag	Cisco IOS XR Release 7.9.1	The telemetry bag, which maintains the collection of telemetry data, needs to be of an ideal size to stream the data effectively. In this release, the optical telemetry bag size has been optimized by introducing the following exclusive sensor paths for OTS and OTS-OCH controllers: • Cisco-IOS-XR-controller-ncs1001-ots-och-oper: ots-och-oper/ots-och-ports /ots-och-port/ots-och-info • Cisco-IOS-XR-controller-ncs1001-ots-oper:ots-oper/ ots-ports/ots-port/ots-info Unlike in previous release, where OTS and OTS-OCH controller's telemetry data was accessed from a single telemetry bag, it is now separated out by the respective sensor paths. The already existing optics-info sensor path now includes only the data of the OPTICS controller. The result is a smaller bag size. Two new native NETCONF yang models are introduced to support the two new sensor paths.

Following is the list of supported configuration YANG models for NCS 1001:

Configuration YANG Models
Cisco-IOS-XR-ncs1001-ots-cfg.yang
Cisco-IOS-XR-ifmgr-cfg.yang
Cisco-IOS-XR-controller-optics-cfg.yang
Cisco-IOS-XR-pmengine-cfg.yang

Following is the list of supported operational YANG models for NCS 1001:

Operational Yang Models
Cisco-IOS-XR-controller-optics-oper.yang
Cisco-IOS-XR-pmengine-oper.yang
Cisco-IOS-XR-plat-chas-invmgr-oper.yang
Cisco-IOS-XR-alarmgr-server-oper.yang
Cisco-IOS-XR-ncs1001-ots-oper.yang
Cisco-IOS-XR-controller-ncs1001-ots-och-oper.yang
openconfig-channel-monitor **Note** For R7.9.1 and R7.10.1, *name* and *monitor-port* parameter values cannot be modified, which may result in an error message.

Note To view the Data Models (Native, Unified, OpenConfig) supported in IOS XR platforms and releases, see the Yang Explorer tool. You can explore the data model definitions, locate a specific model, and view the containers and their respective lists, leaves, leaf lists, Xpaths, and much more.

New Telemetry Bag in NCS 1001

From R.7.9.1 onwards, two new sensor paths are introduced to support the two bags of OTS and OTS-OCH controllers in the NCS 1001 platform. This helps separate the parameters of OTS and OTS-OCH controllers, which helps in reducing the size of each telemetry collection. The optics-info sensor path: Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info was used previously to collect data and parameters for optics, which are now collected by the OTS and OTS-OCH controllers together.

The two new sensor paths introduced for OTS and OTS-OCH controllers in NCS 1001 are as given below.

- Cisco-IOS-XR-controller-ncs1001-ots-och-oper:ots-och-oper/ots-och-ports/ots-och-port/ots-och-info
- Cisco-IOS-XR-controller-ncs1001-ots-oper:ots-oper/ots-ports/ots-port/ots-info

The two new operational YANG models which support these two new sensor paths are as given below.

- Cisco-IOS-XR-controller-ncs1001-ots-oper.yang
- Cisco-IOS-XR-controller-ncs1001-ots-och-oper.yang

Note

- The total size resulting from the contributions of the three SPs (optics-Info, ots-info, and ots-och-info) is now less than 300 KB for a single data collection resulting in 90 percent reduction.

Configure Amplifier Module

Use the Cisco-IOS-XR-ncs1001-ots-cfg.yang YANG model to configure the grid mode parameter of the amplifier module as 100GHz or 50GHz.

Parameter	Example
grid-mode	<?xml version="1.0"?> <rpc message-id="102" 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"> <hardware-module xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ncs1001-ots-cfg"> <node> <location>0_RP0_CPU0</location> <slot> <slot-id>0x2</slot-id> <amplifier> <grid-mode >100g-hz</grid-mode> </amplifier> </slot> </node> </hardware-module> </config> </edit-config> </rpc>

Remove Amplifier Configuration

Use the Cisco-IOS-XR-ncs1001-ots-cfg.yang YANG model to remove the amplifier configuration.

Parameter	Example
grid-mode	<?xml version="1.0"?> <rpc message-id="102" 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"> <hardware-module xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ncs1001-ots-cfg"> <node> <location>0_RP0_CPU0</location> <slot> <slot-id>0x2</slot-id> <amplifier> <grid-mode xc:operation = "delete">100g-hz</grid-mode> </amplifier> </slot>

Parameter	Example
	</node> </hardware-module> </config> </edit-config> </rpc>

Configure Protection Switching Module

Use the Cisco-IOS-XR-ncs1001-ots-cfg.yang YANG model to configure the protection switching module.

Parameter	Example
lockout-from	<?xml version="1.0"?> <rpc message-id="102" 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"> <hardware-module xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ncs1001-ots-cfg"> <node> <location>0_RP0_CPU0</location> <slot> <slot-id>0x3</slot-id> <psm> <lockout-from>protected</lockout-from> </psm> </slot> </node> </hardware-module> </config> </edit-config> </rpc>

Remove Protection Switching Module Configuration

Use the Cisco-IOS-XR-ncs1001-ots-cfg.yang YANG model to remove the protection switching module configuration.

Parameter	Example
lockout-from	<?xml version="1.0"?> <rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target> <candidate/>

Parameter	Example
	</target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <hardware-module xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ncs1001-ots-cfg"> <node> <location>0_RP0_CPU0</location> <slot> <slot-id>0x3</slot-id> <psm> <lockout-from xc:operation = "delete">protected</lockout-from> </psm> </slot> </node> </hardware-module> </config> </edit-config> </rpc>

Configure OTS Controller

Use the Cisco-IOS-XR-controller-optics-cfg.yang YANG model to configure the parameters of the OTS controller such as ampli-control-mode, ampli-channel-power, ampli-gain, ampli-gain-range, ampli-tilt, channel-power-max-delta, osri, safety-control-mode, rx-low-threshold, tx-low-threshold, rx-voa-attenuation, and tx-voa-attenuation.

Parameter	Description
ampli-control-mode	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-amplifier-control-mode>automatic</optics-ots-amplifier-control-mode> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
ampli-channel-power	<?xml version="1.0"?> <rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config>

Parameter	Description
	<target> <candidate/> </target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-amplifier-channel-power>-200</optics-ots-amplifier-channel-power> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
ampli-gain	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-amplifier-gain>100</optics-ots-amplifier-gain> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
ampli-gain-range	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-amplifier-gain-range>normal</optics-ots-amplifier-gain-range> </optics> </interface-configuration>

Parameter	Description
	</interface-configurations> </config> </edit-config> </rpc>
ampli-tilt	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-amplifier-tilt>-35</optics-ots-amplifier-tilt> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
channel-power-max-delta	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-channel-power-max-delta>100</optics-ots-channel-power-max-delta> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
osri	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">

Parameter	Description
	<interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-osri>true</optics-ots-osri> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
safety-control-mode	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-safety-control-mode>auto</optics-ots-safety-control-mode> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
rx-low-threshold	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <rx-thresholds> <rx-threshold> <rx-threshold-type>low</rx-threshold-type> <rx-threshold>100</rx-threshold> </rx-threshold> </rx-thresholds> </optics> </interface-configuration> </interface-configurations> </config>

Parameter	Description
	</edit-config> </rpc>
tx-low-threshold	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/2/0/0</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <tx-thresholds> <tx-threshold> <tx-threshold-type>low</tx-threshold-type> <tx-threshold>100</tx-threshold> </tx-threshold> </tx-thresholds> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
rx-voa-attenuation	<?xml version="1.0"?> <rpc message-id="102" 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"> <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/3/0/1</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-rx-voa-attenuation>112</optics-ots-rx-voa-attenuation> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>
tx-voa-attenuation	<?xml version="1.0"?> <rpc message-id="102" 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">

Parameter	Description
	<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"> <interface-configuration> <active>act</active> <interface-name>Ots0/3/0/1</interface-name> <optics xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-optics-cfg"> <optics-ots-tx-voa-attenuation>135</optics-ots-tx-voa-attenuation> </optics> </interface-configuration> </interface-configurations> </config> </edit-config> </rpc>

Display Parameters of OTS Controllers

Use the Cisco-IOS-XR-controller-ncs1001-ots-oper.yang to display the parameters of OTS controllers. (For example: In case of port 0 of EDFA which is equipped in slot number 2).

Parameter	Description
show controllers ots 0/2/0/0	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:dbff986e-62fb-41e9-aec5-cef6162150bb"> <nc:get> <nc:filter> <ots-oper xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-ncs1001-ots-oper"> <ots-ports> <ots-port> <name>Ots0/2/0/0</name> </ots-port> </ots-ports> </ots-oper> </nc:filter> </nc:get> </nc:rpc>

Display Parameters of OTS OCH Controllers

Use the Cisco-IOS-XR-controller-ncs1001-ots-och-oper.yang to display the parameters of OTS OCH controllers. (For example: In case of port 0 channel 1 of EDFA is equipped in slot 2).

Parameter	Description
show controllers ots-och 0/2/0/0/1	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:f0e8cc35-d8bc-4550-9c6a-7458e6d2af14"><nc:get> <nc:filter> <ots-och-oper xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-controller-ncs1001-ots-och-oper"> <ots-och-ports> <ots-och-port> <name>Ots-Och0/2/0/0/1</name> <ots-och-info> <lane-data/> </ots-och-info> </ots-och-port> </ots-och-ports> </ots-och-oper> </nc:filter> </nc:get> </nc:rpc>

View PM Parameters for OTS Controllers

Use the Cisco-IOS-XR-pmengine-oper.yang YANG model to view the performance monitoring parameters for OTS controllers.

Parameter	Example
show controllers ots 0/2/0/0 pm current 15-min optics 1	<?xml version="1.0" ?> <rpc message-id="8566" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <get> <filter type="subtree"> <performance-management xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-pmengine-oper"> <optics> <optics-ports> <optics-port> <optics-current> <optics-minute15> <optics-minute15-optics> <optics-minute15-optic> <number>1</number> </optics-minute15-optic> </optics-minute15-optics> </optics-minute15> </optics-current> <name>Ots0/2/0/0</name> </optics-port> </optics-ports> </optics> </performance-management> </filter>

Parameter	Example
	</get> </rpc>

OC Support for LLDP on Management Port

Enable LLDP on a Specific Management Interface

Table 2: Feature History

Feature Name	Release	Description
OC (Open Configuration) support for LLDP (Link Layer Discovery Protocol) on Management Port	Cisco IOS XR Release 7.3.1	Open Configuration support for configuring LLDP on management port is available. RPC (Remote Procedure Call) Request and Response messages are used to configure LLDP on management port, which is automated using scripts. This makes the task, less time-consuming.

You can enable LLDP on a specific management interface such as MgmtEth0/RP0/CPU0/1 using the following code:

Edit-Config	
RPC Request	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:a3d3c1e9-1cd1-4348-b7ea-4349d1cfcc32"><nc:edit-config> <nc:target> <nc:candidate/> </nc:target> <nc:config> <lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html" <interfaces> <interface> <name>MgmtEth0/RP0/CPU0/1</name> <config> <name>MgmtEth0/RP0/CPU0/1</name> <enabled nc:operation="create">true</enabled>

Edit-Config	
	</config> </interface> </interfaces> </lldp> </nc:config> </nc:edit-config> </nc:rpc>

Edit-Config	
RPC Response	<?xml version="1.0" ?> <rpc-reply message-id="urn:uuid:a3d3c1e9-1cd1-4348-b7ea-4349d1cfcc32" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>

Disable LLDP on a Specific Management Interface

You can disable LLDP on a specific management interface such as MgmtEth0/RP0/CPU0/1 using the following code:

Edit-Config	
RPC Request	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:3d076d9b-2fa9-4930-9f0c-7cbdc463a188"><nc:edit-config> <nc:target> <nc:candidate/> </nc:target> <nc:config> <lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html"> <interfaces> <interface> <name>MgmtEth0/RP0/CPU0/1</name> <config> <name>MgmtEth0/RP0/CPU0/1</name> <enabled nc:operation="delete">true</enabled>

Edit-Config	
	</config> </interface> </interfaces> </lldp> </nc:config> </nc:edit-config> </nc:rpc>

Edit-Config	
RPC Response	<?xml version="1.0" ?> <rpc-reply message-id="urn:uuid:3d076d9b-2fa9-4930-9f0c-7cbdc463a188" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>

Enable LLDP Globally

You can enable LLDP globally using the following code:

Edit-Config	
RPC Request	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:a4238d46-3891-48e0-b819-a8f31067d251"> <nc:edit-config> <nc:target> <nc:candidate/> </nc:target> <nc:config> <lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html"> <config> <enabled nc:operation="create">true</enabled> </config> </lldp>

Edit-Config	
	</nc:config> </nc:edit-config> </nc:rpc>

Edit-Config	
RPC Response	<?XML version="1.0" ?> <rpc-reply message-id="urn:uuid:a4238d46-3891-48e0-b819-a8f31067d251" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>

Disable LLDP Globally

You can disable LLDP globally using the following code:

Edit-Config	
RPC Request	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:a4238d46-3891-48e0-b819-a8f31067d251"> <nc:edit-config> <nc:target> <nc:candidate/> </nc:target> <nc:config> <lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html"> <config> <enabled nc:operation="remove">true</enabled> </config> </lldp> </nc:config> </nc:edit-config> </nc:rpc>

Edit-Config	
RPC Response	<?xml version="1.0" ?> <rpc-reply message-id="urn:uuid:a4238d46-3891-48e0-b819-a8f31067d251" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>

Get LLDP Configuration

You can get the LLDP configuration using the following code:

Get-Config	
RPC Request	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:f9a226c6-b7d1-4d24-bf76-72b5086a0f35"> <nc:get-config> <nc:source> <nc:running/> </nc:source> <nc:filter> <lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html"> <config> <enabled>true</enabled> </config> </lldp> </nc:filter> </nc:get-config> </nc:rpc>

Get-Config	
RPC Response	<?xml version="1.0" ?> <rpc-reply message-id="urn:uuid:f9a226c6-b7d1-4d24-bf76-72b5086a0f35" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <data>

Get-Config	
	<lldp xmlns="http://ops.openconfig.net/branches/models/master/docs/openconfig-lldp.html"> <config> <enabled>true</enabled> <hello-timer>30</hello-timer> </config> </lldp> </data> </rpc-reply>

Support for Netconf for Read, Write, Execute or Administrative Commands

Table 3: Feature History

Feature Name	Release	Description
NETCONF Support for READ, WRITE, and Execute or Administrative Commands.	Cisco IOS XR Release 7.3.1	Support for IPv4 and IPv6 Ping test using the Cisco-IOS-XR-ping-act YANG model, instead of using CLI commands, is available. RPC (Remote Procedure Call) Request and Response messages are used to do the ping test, which is automated using scripts. This enables you to perform the ping test in a less time-consuming manner and to enhance network scalability.

IPv4 PING Over NETCONF

Use the Cisco-IOS-XR-ping-act YANG model to do the ping test to the destination IPv4 addresses.The following example shows the RPC request and RPC response messages for a successful ping test. The destination host is reachable and the success rate is 100%.

YANG Model	Example
Cisco-IOS-XR-ping-act.yang	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:28170002-365f-45be-a8e1-e1f54d8b64b5"><ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">

YANG Model	Example
	<destination> <destination>10.127.60.1</destination> </destination> </ping> </nc:rpc> <rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:28170002-365f-45be-a8e1-e1f54d8b64b5"> <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <ipv4> <destination>10.127.60.1</destination> <data-size>100</data-size> <timeout>2</timeout> <pattern>abcd</pattern> <rotate-pattern>false</rotate-pattern> <replies> <reply> <reply-index>1</reply-index> <result>!</result> </reply> <reply> <reply-index>2</reply-index> <result>!</result> </reply> <reply> <reply-index>3</reply-index> <result>!</result> </reply> <reply> <reply-index>4</reply-index> <result>!</result> </reply> <reply> <reply-index>5</reply-index>

YANG Model	Example
	<result>!</result> </reply> </replies> <hits>5</hits> <total>5</total> <success-rate>100</success-rate> <rtt-min>1</rtt-min> <rtt-avg>1</rtt-avg> <rtt-max>2</rtt-max> </ipv4> </ping-response> </rpc-reply>

The following example shows the RPC request and RPC response messages for a failure ping test. The destination host is not reachable and the success rate is 0%.

YANG model	Example
Cisco-IOS-XR-ping-act.yang	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:28170002-365f-45be-a8e1-e1f54d8b64b5"><ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <destination> <destination>10.127.60.1</destination> </destination> </ping> </nc:rpc> <rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:02800209-6ebf-4955-8588-f6cdfd6f2750"> <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <ipv4> <destination>10.127.60.171</destination> <data-size>100</data-size> <timeout>2</timeout> <pattern>abcd</pattern> <rotate-pattern>false</rotate-pattern> <replies>

YANG model	Example
	<reply> <reply-index>1</reply-index> <result>.</result> </reply> <reply> <reply-index>2</reply-index> <result>.</result> </reply> <reply> <reply-index>3</reply-index> <result>.</result> </reply> <reply> <reply-index>4</reply-index> <result>.</result> </reply> <reply> <reply-index>5</reply-index> <result>.</result> </reply> </replies> <hits>0</hits> <total>5</total> <success-rate>0</success-rate> </ipv4> </ping-response> </rpc-reply>

Note In the above examples, 10.127.60.1 is the IPv4 address of the node.

IPv6 PING Over NETCONF

Before you begin

Table 4: Feature History

Feature Name	Release	Description
NETCONF Support for READ, WRITE, and Execute or Administrative Commands.	Cisco IOS XR Release 7.3.1	Support for IPv4 and IPv6 Ping test using the Cisco-IOS-XR-ping-act YANG model, instead of using CLI commands, is available. RPC (Remote Procedure Call) Request and Response messages are used to do the ping test, which is automated using scripts. This enables you to perform the ping test in a less time-consuming manner and to enhance network scalability.

Use the Cisco-IOS-XR-ping-act YANG model to do the ping test to the destination IPv6 addresses. The following example shows the RPC request and RPC response messages for a successful ping test. The destination host is reachable and the success rate is 100%.

YANG model	Example
Cisco-IOS-XR-ping-act.yang	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:28170002-365f-45be-a8e1-e1f54d8b64b5"><ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <destination> <destination>2001:420:5446:2014::281:178</destination> </destination> </ping> </nc:rpc> <rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:15798adc-f9f9-41b2-9aa5-a1c88dd788e8"> <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <ipv6> <destination>2001:420:5446:2014::281:178</destination> <repeat-count>50</repeat-count> <data-size>100</data-size> <timeout>2</timeout> <pattern>abcd</pattern>

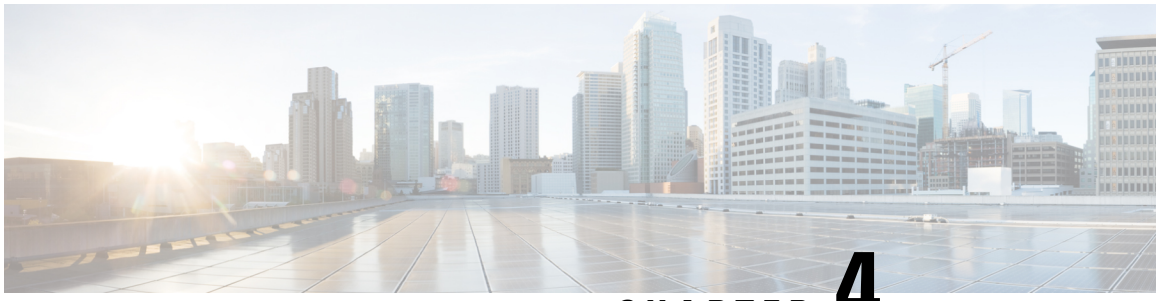
YANG model	Example
	<rotate-pattern>false</rotate-pattern> <replies> <reply> <reply-index>1</reply-index> <result>!</result> </reply> <reply> <reply-index>2</reply-index> <result>!</result> </reply> <reply> <reply-index>3</reply-index> <result>!</result> </reply> <reply> <reply-index>4</reply-index> <result>!</result> </reply> <reply> <reply-index>5</reply-index> <result>!</result> </reply> </replies> <hits>5</hits> <total>5</total> <success-rate>100</success-rate> <rtt-min>1</rtt-min> <rtt-avg>1</rtt-avg> <rtt-max>2</rtt-max> </ipv6> </ping-response> </rpc-reply>

The following example shows the RPC request and RPC response messages for a failure ping test. The destination host is not reachable and the success rate is 0%.

YANG model	Example
Cisco-IOS-XR-ping-act.yang	<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:28170002-365f-45be-a8e1-e1f54d8b64b5"><ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <destination> <destination>2001:420:5446:2014::281:178</destination> </destination> </ping> </nc:rpc> <rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:02800209-6ebf-4955-8588-f6cdfd6f2750"> <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act"> <ipv6> <destination>2001:420:5446:2014::281:178</destination> <data-size>100</data-size> <timeout>2</timeout> <pattern>abcd</pattern> <replies> <reply> <reply-index>1</reply-index> <result>.</result> </reply> <reply> <reply-index>2</reply-index> <result>.</result> </reply> <reply> <reply-index>3</reply-index> <result>.</result> </reply> <reply> <reply-index>4</reply-index> <result>.</result>

YANG model	Example
	</reply> <reply> <reply-index>5</reply-index> <result>.</result> </reply> </replies> <hits>0</hits> <total>5</total> <success-rate>0</success-rate> </ipv6> </ping-response> </rpc-reply>

Note In the above examples, 2001:420:5446:2014::281:178 is the IPv6 address of the node.



CHAPTER 4

Configuring NCS 1001 Using OpenConfig Data Model

Openconfig is a working group of network operators which defines a set of vendor-neutral YANG data models supporting various network functions and devices.

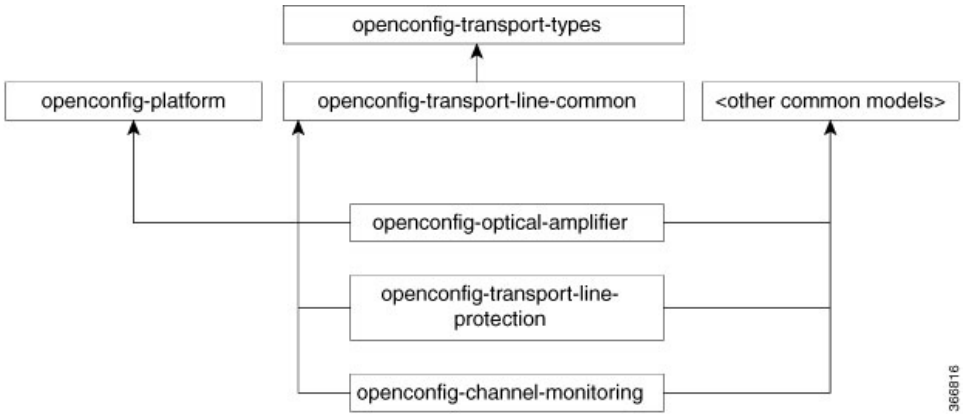
A complete list of supported open configuration models is available at https://github.com/openconfig/public/tree/master/release/models/optical-transport.

Cisco NCS1001 supports openconfig models according to the optical transport functions available on system. The following are the openconfig supported models:

- Amplifier model, supported by NCS1001 EDFA modules
- Transport Line Protection model, supported by NCS1001 PSM modules
- Channel Monitoring model, supported by NCS1001 EDFA by means of its OCM capability.

The Openconfig platform model is a common model where all the devices are listed. The openconfig model hierarchy is as follows:

Figure 5: Hierarchy of Cisco NCS 1001



The Openconfig platform model retrieves inventory information and it's instantiation on Cisco NCS1001 contains the models defined in the following sections.

- Openconfig Optical Amplifier Model, on page 40
- Openconfig Protection Switching Model, on page 41

• Openconfig Channel Monitoring Model, on page 42

Openconfig Optical Amplifier Model

The following is the tree structure of Openconfig Optical Amplifier model:

```
module: openconfig-optical-amplifier
   +--rw optical-amplifier
      +--rw amplifiers
      |  +--rw amplifier* [name]
      |     +--rw name      -> ../config/name
      |     +--rw config
      |     |  +--rw name?               string
      |     |  +--rw target-gain?        decimal64
      |     |  +--rw target-gain-tilt?   decimal64
      |     |  +--rw enabled?            boolean
      |     +--ro state
      |        +--ro name?                   string
      |        +--ro type?                   identityref
      |        +--ro target-gain?            decimal64
      |        +--ro target-gain-tilt?       decimal64
      |        +--ro gain-range?             identityref
      |        +--ro amp-mode?               identityref
      |        +--ro enabled?                boolean
      |        +--ro ingress-port?           -> /oc-platform:components/component/name
      |        +--ro egress-port?            -> /oc-platform:components/component/name
      |        +--ro actual-gain
      |        |  +--ro instant?   decimal64
      |        +--ro actual-gain-tilt
      |        |  +--ro instant?   decimal64
      |        +--ro input-power-total
      |        +--ro input-power-c-band
      |        |  +--ro instant?   decimal64
      |        |  +--ro avg?       decimal64
      |        |  +--ro min?       decimal64
      |        |  +--ro max?       decimal64
      |        +--ro input-power-l-band
      |        +--ro output-power-total
      |        +--ro output-power-c-band
      |        |  +--ro instant?   decimal64
      |        |  +--ro avg?       decimal64
      |        |  +--ro min?       decimal64
      |        |  +--ro max?       decimal64
      |        +--ro output-power-l-band
      |        +--ro laser-bias-current
      |        +--ro optical-return-loss
      +--rw supervisory-channels
         +--rw supervisory-channel* [interface]
            +--rw interface    -> ../config/interface
            +--rw config
            +--ro state
               +--ro input-power
               |  +--ro instant?   decimal64
               |  +--ro avg?       decimal64
               |  +--ro min?       decimal64
               |  +--ro max?       decimal64
               +--ro output-power
               |  +--ro instant?   decimal64
               |  +--ro avg?       decimal64
               |  +--ro min?       decimal64
               |  +--ro max?       decimal64
               +--ro laser-bias-current
```

Inventory Details

Each EDFA module pluggable within Cisco NCS1001 slot contains two optical amplifiers such as a booster and a pre-amplifier. The list of components applicable as amplifier name is listed in the following table:

Component	Naming Convention
EDFA Module, Pre-Amplifier	0/S-AMP-PRE
EDFA Module, Booster Amplifier	0/S-AMP-BST

Openconfig Protection Switching Model

The following is the tree structure of Transport Line Protection model:

```
module: openconfig-transport-line-protection
   +--rw aps
      +--rw aps-modules
         +--rw aps-module* [name]
            +--rw name      -> ../config/name
            +--rw config
            |  +--rw name?                         -> /oc-platform:components/component/name

            |  +--rw primary-switch-threshold?     decimal64
            |  +--rw secondary-switch-threshold?   decimal64
            +--ro state
            |  +--ro name?                         -> /oc-platform:components/component/name

            |  +--ro primary-switch-threshold?     decimal64
            |  +--ro secondary-switch-threshold?   decimal64
            |  +--ro active-path?                  identityref
            +--rw ports
               +--rw line-primary-in
               |  +--rw config
               |  |  +--rw target-attenuation?   decimal64
               |  +--ro state
               |     +--ro target-attenuation?   decimal64
               |     +--ro attenuation?          decimal64
               |     +--ro optical-power
               |        +--ro instant?   decimal64
               |        +--ro avg?       decimal64
               |        +--ro min?       decimal64
               |        +--ro max?       decimal64
               +--rw line-primary-out
               |  +--rw config
               |  |  +--rw target-attenuation?   decimal64
               |  +--ro state
               |     +--ro target-attenuation?   decimal64
               |     +--ro attenuation?          decimal64
               |     +--ro optical-power
               |        +--ro instant?   decimal64
               |        +--ro avg?       decimal64
               |        +--ro min?       decimal64
               |        +--ro max?       decimal64
               +--rw line-secondary-in
               |  +--rw config
               |  |  +--rw target-attenuation?   decimal64
               |  +--ro state
               |     +--ro target-attenuation?   decimal64
```

```
|     +--ro attenuation?          decimal64
|     +--ro optical-power
|        +--ro instant?   decimal64
|        +--ro avg?       decimal64
|        +--ro min?       decimal64
|        +--ro max?       decimal64
+--rw line-secondary-out
|  +--rw config
|  |  +--rw target-attenuation?   decimal64
|  +--ro state
|     +--ro target-attenuation?   decimal64
|     +--ro attenuation?          decimal64
|     +--ro optical-power
|        +--ro instant?   decimal64
|        +--ro avg?       decimal64
|        +--ro min?       decimal64
|        +--ro max?       decimal64
+--rw common-in
|  +--rw config
|  +--ro state
|     +--ro optical-power
+--rw common-output
   +--rw config
   +--ro state
      +--ro optical-power
         +--ro instant?   decimal64
         +--ro avg?       decimal64
         +--ro min?       decimal64
         +--ro max?       decimal64
```

Inventory Details

The Cisco NCS1001 PSM pluggable module is provided through the following inventory information:

Component	Naming Convention
EDFA Module, PSM	0/S-PSM-OM

Openconfig Channel Monitoring Model

The following is the tree structure of OpenConfig Channel Monitor model:

```
module: openconfig-channel-monitor
   +--rw channel-monitors
      +--rw channel-monitor* [name]
         +--rw name        -> ../config/name
         +--rw config
         |  +--rw name?           -> /oc-platform:components/component/name
         |  +--rw monitor-port?   -> /oc-platform:components/component/name
         +--ro state
         |  +--ro name?           -> /oc-platform:components/component/name
         |  +--ro monitor-port?   -> /oc-platform:components/component/name
         +--rw channels
            +--ro channel* [lower-frequency upper-frequency]
               +--ro lower-frequency    -> ../state/lower-frequency
               +--ro upper-frequency    -> ../state/upper-frequency
               +--ro state
                  +--ro lower-frequency?   oc-opt-types:frequency-type
```

```
+--ro upper-frequency?   oc-opt-types:frequency-type
+--ro psd?               oc-types:ieeefloat32
```

Inventory Details

Channel monitoring functions are provided through OCM embedded on the EDFA pluggable modules. The following inventory items provide the names of the channel monitoring:

Component	Naming Convention
EDFA Module, Pre-Amplifier OCM	0/S-CHMON-PRE
EDFA Module, Booster Amplifier OCM	0/S-CHMON-BST