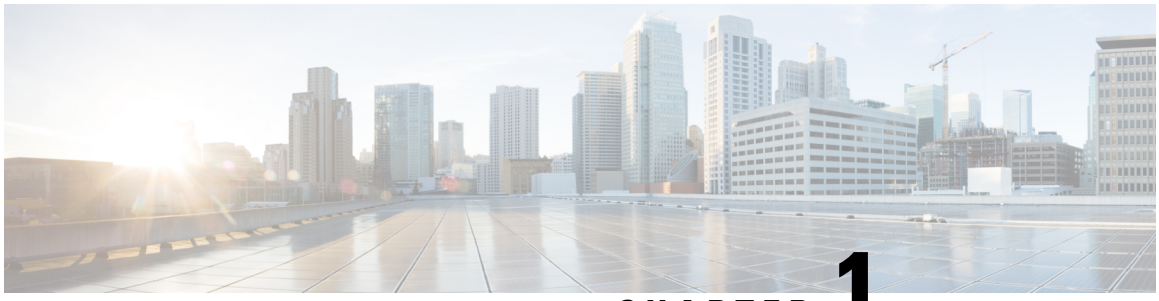
Data Models Configuration Guide for Cisco NCS 1014, IOS XR Releases 7.11.x and 24.1.x

First Published: 2023-11-30

Last Modified: 2024-03-21

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 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 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

Name	Description
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

Data Model and CLI Comparison

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

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

The options available using the CLI are defined as leaf-nodes in data models. The defined data types, indicated corresponding to each leaf-node, help the user to understand the required inputs.

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.

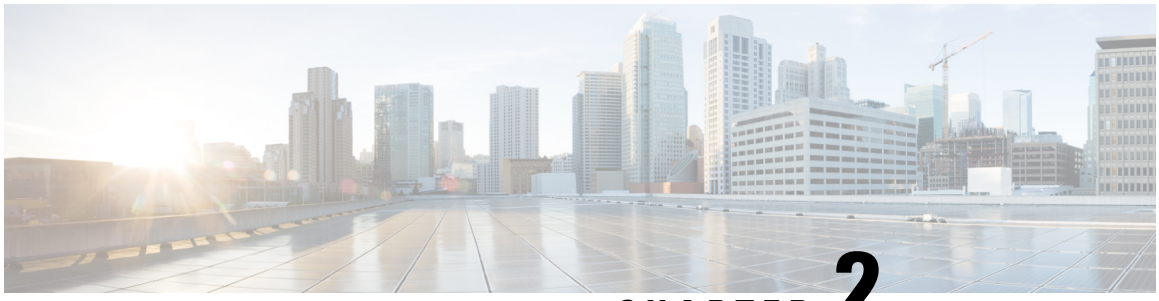
gRPC requests can be encoded and sent across to the router using JSON. gRPC IDL also supports the exchange of CLI.

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

Use Data Models

Figure 1: 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:

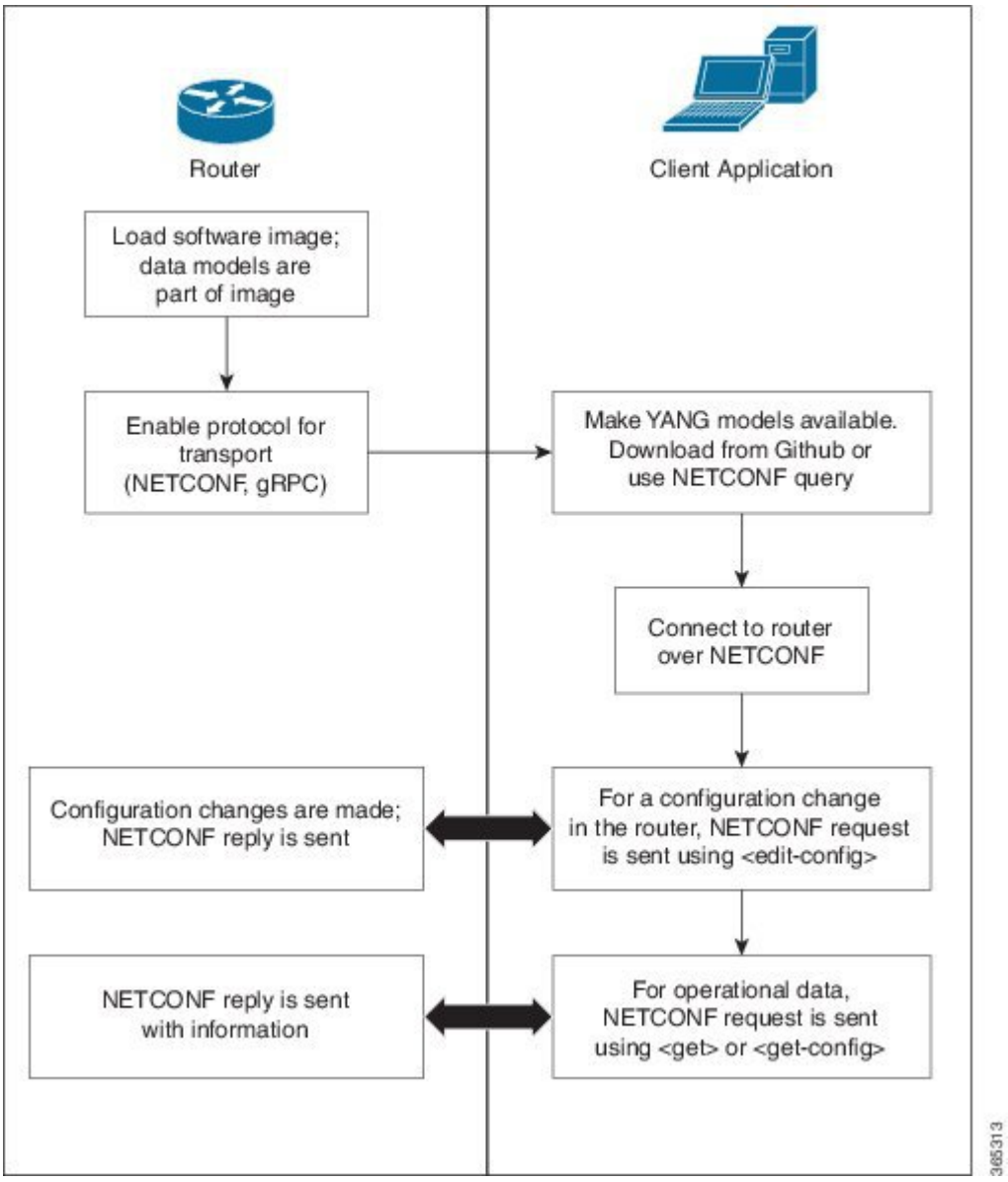
```
https://github.com/YangModels/yang/tree/master/vendor/cisco/xr
```

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 Netconf over SSH. Enable Netconf on the router on a suitable port.
3. From the client application, connect to the router using Netconf over SSH. Run Netconf operations to make configuration changes or get operational data.

Figure 2: 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

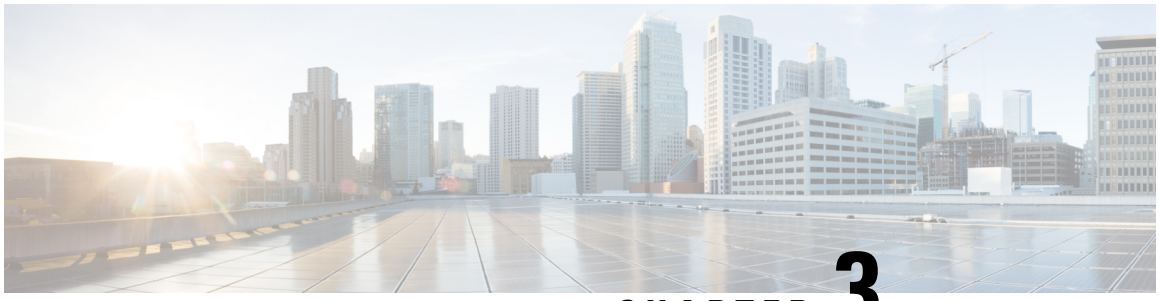
Use the following procedure to enable gRPC over HTTPS/2. gRPC supports both, the IPv4 and IPv6 address families (default is IPv4).

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

Supported YANG Models in NCS 1014

Supported Yang Models

The following is the list of supported config, oper, and act YANG models for NCS 1014:

Table 1: Native Models

Config Models	Oper Models
Cisco-IOS-XR-osa-linesystem-cfg.yang	Cisco-IOS-XR-osa-hwmod-linesys-oper.yang
Cisco-IOS-XR-controller-ots-cfg.yang	Cisco-IOS-XR-controller-ots-oper.yang
Cisco-IOS-XR-ots-och-cfg.yang	Cisco-IOS-XR-controller-ots-och-oper.yang
Cisco-IOS-XR-controller-oms-cfg	Cisco-IOS-XR-controller-oms-oper.yang
Cisco-IOS-XR-controller-och-cfg	Cisco-IOS-XR-controller-och-oper.yang
Cisco-IOS-XR-controller-osc-cfg.yang	Cisco-IOS-XR-controller-osc-oper.yang
Cisco-IOS-XR-controller-dfb-cfg.yang	Cisco-IOS-XR-controller-dfb-oper.yang
Cisco-IOS-XR-pmengine-cfg.yang	Cisco-IOS-XR-pmengine-oper.yang
Cisco-IOS-XR-olc-cfg.yang	Cisco-IOS-XR-olc-oper.yang
Cisco-IOS-XR-fpd-infra-cfg.yang	
Cisco-IOS-XR-osa-ct-cfg.yang	Cisco-IOS-XR-osa-controller-optics-oper.yang
Cisco-IOS-XR-osa-sp-cfg.yang	Cisco-IOS-XR-osa-hwmod-linesys-oper.yang
Cisco-IOS-XR-osa-cfg.yang	Cisco-IOS-XR-osa-oper.yang
Cisco-IOS-XR-ikev2-cfg.yang	Cisco-IOS-XR-ikev2-oper.yang
Cisco-IOS-XR-controller-optics-cfg.yang	Cisco-IOS-XR-controller-optics-oper.yang
Cisco-IOS-XR-ethernet-lldp-cfg.yang	Cisco-IOS-XR-ethernet-lldp-oper.yang
Cisco-IOS-XR-telemetry-model-driven-cfg.yang	Cisco-IOS-XR-telemetry-model-driven-oper.yang

Config Models	Oper Models
Cisco-IOS-XR-ifmgr-cfg.yang	Cisco-IOS-XR-envmon-oper.yang
Cisco-IOS-XR-fpd-infra-cfg.yang	Cisco-IOS-XR-show-fpd-loc-ng-oper.yang
Cisco-IOS-XR-invproxy-hwmodule-cfg.yang	Cisco-IOS-XR-procfind-oper.yang
Cisco-IOS-XR-syncc-controller-cfg.yang	Cisco-IOS-XR-procthreadname-oper.yang
Cisco-IOS-XR-drivers-media-eth-gl-pfc-wd-cfg.yang	Cisco-IOS-XR-invmgr-oper.yang
Cisco-IOS-XR-drivers-media-eth-cfg.yang	Cisco-IOS-XR-plat-chas-invmgr-ng-oper.yang
Cisco-IOS-XR-drivers-icpe-ethernet-cfg.yang	Cisco-IOS-XR-platform-oper.yang
Cisco-IOS-XR-drivers-media-eth-cfg.yang	Cisco-IOS-XR-invmgr-diag-oper.yang
Cisco-IOS-XR-drivers-media-eth-gl-pfc-wd-cfg.yang	Cisco-IOS-XR-nto-misc-oper.yang
Cisco-IOS-XR-sysmgr-cfg.yang	Cisco-IOS-XR-wd-oper.yang
Cisco-IOS-XR-um-ncs-hw-module-osa-cfg.yang	Cisco-IOS-XR-ledmgr-oper.yang
	Cisco-IOS-XR-shellutil-filesystem-oper.yang
	Cisco-IOS-XR-shellutil-oper.yang
	Cisco-IOS-XR-drivers-media-eth-oper.yang
	Cisco-IOS-XR-mediasvr-linux-oper.yang
	Cisco-IOS-XR-drivers-media-eth-oper.yang
	Cisco-IOS-XR-sysmgr-oper.yang
	Cisco-IOS-XR-procmem-oper.yang
	Cisco-IOS-XR-procfind-oper.yang
Act Models	
Cisco-IOS-XR-upgrade-fpd-ng-act.yang	
Cisco-IOS-XR-shellutil-delete-act.yang	
Cisco-IOS-XR-drivers-media-eth-act.yang	
Cisco-IOS-XR-shellutil-copy-act.yang	
Cisco-IOS-XR-shellutil-copy-act.yang	
Cisco-IOS-XR-drivers-media-eth-act.yang	
Cisco-IOS-XR-sysmgr-act.yang	
Cisco-IOS-XR-system-reboot-act	
Cisco-IOS-XR-install-act.yang	

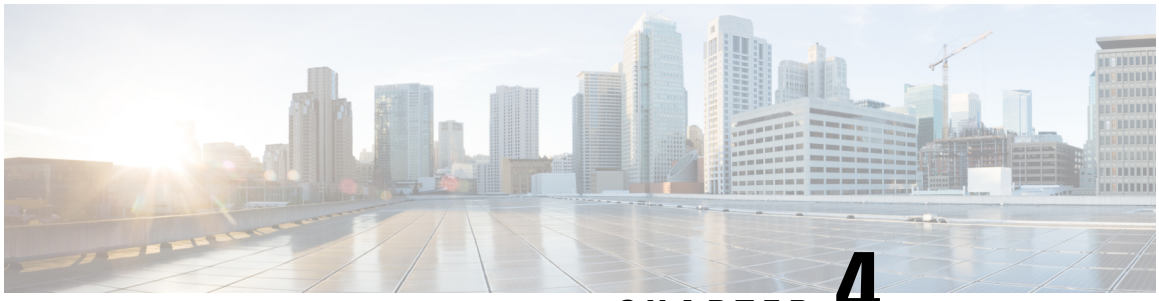
Config Models	Oper Models
Cisco-IOS-XR-install-augmented-act	

The following is the list of supported Open Config models:

Table 2: OpenConfig Models

openconfig-platform.yang
openconfig-platform-transceiver.yang
openconfig-terminal-device.yang
openconfig-interfaces.yang
openconfig-system.yang
openconfig-network-instance
openconfig-procmon.yang

See https://cfnng.cisco.com/ios-xr/yang-explorer/view-data-model for the list of Yang models supported by NCS 1014.



CHAPTER 4

OpenConfig Support for NCS1K14-2.4T-K9 Card

The NCS1K14-2.4T-K9 card is a single slot line card. The card is equipped with six QSFPDD and two CIM-8 ports. This chapter briefs the detail configurations, client and trunk optics, supported OpenConfig models for the NCS1K14-2.4T-K9 card.

Overview

The NCS1K14-2.4T-K9 card is a single slot line card. The card is equipped with six QSFPDD and two CIM-8 ports. You can configure six QSFPDD ports as client and two CIM-8 as trunk.

The NCS1K14-2.4T-K9 card supports both transponder (TXP) and muxponder(MXP) configuration and they can coexist on the same line card.

Supported Operational modes, Optics, and OpenConfig Models

The NCS1K14-2.4T-K9 card supports the following Operational modes, client and trunk optics, and OpenConfig models:

Operational Modes

The following table provides information for Operational modes, config in SliceMode, Slice 0 Client, and Slice 1 Client:

Operational modes	Config in SliceMode	Slice 0 Client	Slice 1 Client
400G	4X100GE	1	4
600G	400GE+2x100GE	1,2	4,5
800G	2x400GE	1,2	4,5

Operational modes	Config in SliceMode	Slice 0 Client	Slice 1 Client
1000G	2x400GE+2x100GE	1,2,3	4,5,6

Client Optics

The following table provides information about PIDs, and related interface, transmit power, transmit wavelength, fiber type, fiber connector, and distance support:

PID	Interface	Transmit power	Transmit wavelength	Fiber type	Fiber connector	Distance support	Description
QDD-400G-FR4-S	400GE	-7.0 to +6.0 dbm per wavelength	1310 nm	Duplex SMF	Duplex LC connector	2km	Only be used as 400GE non-breakout mode
QDD-400G-DR4-S	400GBASE-DR4	-10.1 to +4.0 dbm per wavelength	1310 nm	MPO-12 parallel SMF	12-fiber MPO	500m	Can be used as 4x100GE breakout mode
QDD-400G-AOCxM	400GBASE-AOC	-10.1 to +4.0 dbm per wavelength	850 nm	MMF	AOC	1, 2, 3, 5, 7, 10, 15, 20, 25, and 30 meters	Only be used as 400GE non-breakout mode
QDD-4X100G-LR-S	10Base-LR	–8.2 to +0.5 per wavelength	1310nm	G.652 micron SMF	12-fiber MPO	10km	Can be used in 4x100GE breakout mode as well as 400GE non-breakout mode.

Trunk Optics

Note The transceiver name appears in the new format "Optics *rack*/*slot*/*instance*/*port*" from release 7.11.1.

The following table provides information for PIDs, it's related payloads, trunk ports, and inventory details:

PID	Payload	Trunk Port Number	Inventory Details
CIM8-C-K9	400G, 600G, 800G, and 1000G	0, and 7	NAME:"Optics0/1/0/0", DESCR: "Cisco CIM8 C K9 Pluggable Optics Module" PID: CIM8-C-K9, VID:VES1,SN:ACA273401DG

OpenConfig Models

The NCS1K14-2.4T-K9 card supports the following OpenConfig models:

Table 3: Supported OC Models

Model	Feature
openconfig-platform.yang	Inventory and LCMode
openconfig-platform-transceiver.yang	Pluggable Inventory and Operational Data
openconfig-terminal-device.yang	Logical and Optical Channels – Datapath and OperData
openconfig-interface.yang	Optical Interface Enable/Disable (shut/no-shut)
openconfig-system.yang (augumented with openconfig-alarms)	Alarms
openconfig/gnoi/os.proto	Software Upgrade
Openconfig/gnoi/diag.proto	PRBS Testing

Extended Terminal Device Configuration for Baud Rate

The following table provides standard operational-modes for configuring the baud rate:

Table 4: Standard Operational Modes

Mode	FEC	Baud-Rate	Description
4201	SD_15	138.000000	SoftDecision_FEC15:Baud_138.00000000
4202	SD_15	139.000000	SoftDecision_FEC15:Baud_139.00000000
4203	SD_15	140.000000	SoftDecision_FEC15:Baud_140.00000000
4204	SD_15	141.000000	SoftDecision_FEC15:Baud_141.00000000
4205	SD_15	142.000000	SoftDecision_FEC15:Baud_142.00000000
4206	SD_15	100.000000	SoftDecision_FEC15:Baud_100.00000000
4207	SD_15	80.000000	SoftDecision_FEC15:Baud_80.00000000

Mode	FEC	Baud-Rate	Description
4208	SD_15	88.000000	SoftDecision_FEC15:Baud_88.00000000
4209	SD_15	98.000000	SoftDecision_FEC15:Baud_98.00000000
4210	SD_15	108.000000	SoftDecision_FEC15:Baud_108.00000000
4211	SD_15	118.000000	SoftDecision_FEC15:Baud_118.00000000
4212	SD_15	128.000000	SoftDecision_FEC15:Baud_128.00000000
4213	SD_15	110.000000	SoftDecision_FEC15:Baud_110.00000000
4214	SD_15	111.000000	SoftDecision_FEC15:Baud_111.00000000
4215	SD_15	112.000000	SoftDecision_FEC15:Baud_112.00000000
4216	SD_15	113.000000	SoftDecision_FEC15:Baud_113.00000000
4217	SD_15	114.000000	SoftDecision_FEC15:Baud_114.00000000
4218	SD_15	115.000000	SoftDecision_FEC15:Baud_115.00000000

You can use the **extended terminal-device baud rate** to set a new baud rate value compared to the value provided in the **Standard Operational Mode** table.

Note The Optical Channel name appears in the new format "OpticalChannel *rack*/*slot*/*instance*/*port*" from release 7.11.1.

Sample Configuration

```
----------------------------------
Edit config baud-rate
----------------------------------
<edit-config>
  <target>
    <candidate/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>

        <name>OpticalChannel0/0/0/0</name>
        <optical-channel xmlns="http://openconfig.net/yang/terminal-device">
        <extended
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-openconfig-terminal-device-ext">
            <config>
             <baud-rate>15.1234567</baud-rate>
            </config>
       </extended>
      </optical-channel>
     </component>
   </components>
  </config>
</edit-config>
```

Note If both the operating mode and extended baud rate exist, the line card employs the extended baud rate value.

Extended Transceiver Model

The extended transceiver model provides you with the Forward Error Correction (FEC) information for individual physical-channels.

Sample Configuration:

```
"Optics0/1/0/8": {
          "openconfig-platform-transceiver:transceiver": {
            "physical-channels": {
              "channel": {
                "1": {
                  "state": {
                    "index": 1,
                    "input-power": {
                      "avg": 1.64,
                      "instant": 1.6,
                      "interval": 10000000000,
                      "max": 1.72,
                      "max-time": 1649788692425519767,
                      "min": 1.59,
                      "min-time": 1649788694425593293
                    },
                    "laser-bias-current": {
                      "avg": 800,
                      "instant": 800,
                      "interval": 10000000000,
                      "max": 800,
                      "max-time": 1649788690426089532,
                      "min": 800,
                      "min-time": 1649788690426089532
                    },
                    "output-frequency": 228849200,
                    "output-power": {
                      "avg": 1.62,
                      "instant": 1.61,
                      "interval": 10000000000,
                      "max": 1.62,
                      "max-time": 1649788690426089532,
                      "min": 1.62,
                      "min-time": 1649788690426089532
                      }
                    },
                   "extended": {
                      "state": {
                         index: 1
                         "fec-mode": "openconfig-platform-types:FEC_ENABLED",
                         "fec-uncorrectable-words": 0,
                         "fec-corrected-words":0
                              }
                  }
                 }
                }
                }
```

```
  }
}
```

Client Configuration Details

The following table explains the different commands that are used for 100G and 400GE client ports.

Table 5: Configuration Details for 100G and 400GE Client Ports

Client Port	Logical Channel	Trunk ODU	Coherent DSP	Optical Channel
100G	"index":101, "rate-class": "openconfig-transport-types: TRIB_RATE_100G", "description": "Client Logical Channel", "admin-state":"ENABLED", "loopback-mode":"NONE", "trib-protocol": "openconfig-transport-types: PROT_100G_MLG", "logical-channel-type": "openconfig-transport-types: PROT_ETHERNET"	index": 111, "config": { "index": 111, "rate-class": "openconfig-transport-types: TRIB_RATE_100G", "admin-state": "ENABLED", "description": "Trunk-side-ODU", "trib-protocol": "openconfig-transport-types: PROT_ODUFLEX_CBR", "logical-channel-type": "openconfig-transport-types: PROT_OTN"	"index": 212, "config": { "index": 212, "admin-state": "ENABLED", "loopback-mode":"NONE", "description": "Coherent DSP", "rate-class": "openconfig-transport-types: TRIB_RATE_400G", "logical-channel-type": "openconfig-transport-types: PROT_OTN"	"name": "OpticalChannel0/1/0/0", "openconfig-terminal-device: optical-channel": {"config": {"frequency": "193100000", "target-output-power": -700,"operational-mode": 4178, "line-port": "Optics0/1/0/0"

Client Port	Logical Channel	Trunk ODU	Coherent DSP	Optical Channel
400GE	"index": 101, "rate-class": "openconfig-transport-types: TRIB_RATE_400G", "description": "Client Logical Channel", "admin-state": "ENABLED", "loopback-mode": "NONE", "trib-protocol": "openconfig-transport-types: PROT_400GE", "logical-channel-type": "openconfig-transport-types: PROT_ETHERNET"	"index": 211, "config": { "index": 211, "rate-class": "openconfig-transport-types: TRIB_RATE_400G", "admin-state": "ENABLED", "description": "Trunk-side-ODU", "trib-protocol": "openconfig-transport-types: PROT_ODUFLEX_CBR", "logical-channel-type": "openconfig-transport-types: PROT_OTN"	"index":212, "config": { "index": 212, "admin-state": "ENABLED", "loopback-mode":"NONE", "description":"Coherent DSP", "rate-class": "openconfig-transport-types: TRIB_RATE_400G", "logical-channel-type": "openconfig-transport-types: PROT_OTN"	"name": "OpticalChannel0/1/0/0", "openconfig-terminal-device: optical-channel": { "config": { "frequency": 193100000", "target-output-power": -700, "line-port": "Optics0/1/0/0"

Note Trunk payload rate determines the Trib rate.

Sample Configurations

Configuring 400 TXP (Client and Slice)

```
{
 "openconfig-terminal-device:terminal-device": {
  "logical-channels": {
   "channel": [
    {
     "index": 101,
     "config": {
      "index": 101,
      "rate-class": "openconfig-transport-types:TRIB_RATE_400G",
      "admin-state": "ENABLED",
      "description": "Client Logical Channel",
      "trib-protocol": "openconfig-transport-types:PROT_400GE",
      "logical-channel-type": "openconfig-transport-types:PROT_ETHERNET"
     },
     "ingress": {
      "config": {

   "transceiver": "Optics0/1/0/1"
      }
```

```
 },
 "logical-channel-assignments": {
  "assignment": [
   {
    "index": 1,
    "config": {
     "index": 1,
     "allocation": "400",
     "assignment-type": "LOGICAL_CHANNEL",
     "description": "logical to logical assignemnt",
     "logical-channel": 111
    }
   }
  ]
 }
},
{
 "index": 111,
 "config": {
  "index": 111,
  "rate-class": "openconfig-transport-types:TRIB_RATE_400G",
  "admin-state": "ENABLED",
  "description": "Trunk-side-ODU",
  "trib-protocol": "openconfig-transport-types:PROT_ODUFLEX_CBR",
  "logical-channel-type": "openconfig-transport-types:PROT_OTN"
 },
 "logical-channel-assignments": {
  "assignment": [
   {
    "index": 1,
    "config": {
     "index": 1,
     "allocation": "400",
     "assignment-type": "LOGICAL_CHANNEL",
     "description": "logical to Logical",
     "logical-channel": 30000
    }
   }
  ]
 }
},
{
 "index": 201,
 "config": {
  "index": 201,
  "rate-class": "openconfig-transport-types:TRIB_RATE_400G",
  "admin-state": "ENABLED",
  "description": "Client Logical Channel",
  "trib-protocol": "openconfig-transport-types:PROT_400GE",
  "logical-channel-type": "openconfig-transport-types:PROT_ETHERNET"
 },
 "ingress": {
  "config": {

   "transceiver": "Optics0/1/0/2"
  }
 },
 "logical-channel-assignments": {
  "assignment": [
   {
    "index": 1,
    "config": {
     "index": 1,
     "allocation": "400",
```

```
            "assignment-type": "LOGICAL_CHANNEL",
            "description": "logical to logical assignemnt",
            "logical-channel": 211
           }
          }
         ]
        }
       },
       {
        "index": 211,
        "config": {
         "index": 211,
         "rate-class": "openconfig-transport-types:TRIB_RATE_400G",
         "admin-state": "ENABLED",
         "description": "Trunk-side-ODU",
         "trib-protocol": "openconfig-transport-types:PROT_ODUFLEX_CBR",
         "logical-channel-type": "openconfig-transport-types:PROT_OTN"
        },
        "logical-channel-assignments": {
         "assignment": [
          {
           "index": 1,
           "config": {
            "index": 1,
            "allocation": "400",
            "assignment-type": "LOGICAL_CHANNEL",
            "description": "logical to Logical",
            "logical-channel": 30000
           }
          }
         ]
        }
       },
       {
        "index": 30000,
        "config": {
         "index": 30000,
         "admin-state": "ENABLED",
         "description": "Coherent DSP",
         "logical-channel-type": "openconfig-transport-types:PROT_OTN"
        },
        "logical-channel-assignments": {
         "assignment": [
          {
           "index": 1,
           "config": {
            "index": 1,
            "allocation": "800",
            "assignment-type": "OPTICAL_CHANNEL",
            "description": "logical to optical",

            "optical-channel": "OpticalChannel0/1/0/0"
           }
          }
         ]
        }
       }
      ]
     }
    },
    "openconfig-platform:components": {
     "component": [
      {
```

```
    "name": "OpticalChannel0/1/0/0",
    "openconfig-terminal-device:optical-channel": {
     "config": {
      "line-port": "Optics0/1/0/0"
     }
    }
   }
  ]
 },
 "openconfig-interfaces:interfaces": {
  "interface": [
   {
    "name": "Optics0/1/0/0",
    "config": {
     "name": "Optics0/1/0/0",
     "type": "iana-if-type:opticalChannel",
     "description": "T0",
     "enabled": "true"
    }
   }
  ]
 }
}
```