Oracle® Communications Network Integrity

NETCONF-Driven Network Discovery and UIM Integration Cartridge Guide

Release 8.0
G34939-01
October 2025

Oracle Communications Network Integrity NETCONF-Driven Network Discovery and UIM Integration Cartridge Guide, Release 8.0

G34939-01

Contents

About This Content

This guide describes the functionality and design of the Oracle Communications NETCONF Network Discovery and UIM Integration cartridge.

Audience

This guide is intended for network administrators who want to understand the design and functionality of this cartridge and for Network Integrity developers who want to either build or extend similar cartridges.

You should have a good working knowledge of Network Configuration Protocol (NETCONF) and its operations, specifications, and the use of Oracle Communications Design Studio for Network Integrity.

You should be familiar with the following documents, included with this release:

- *Oracle Communications Network Integrity Concepts*
- *Oracle Communications Network Integrity Developer's Guide*

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

This chapter provides an overview of the Oracle Communications Network Integrity NETCONF Network Discovery and UIM Integration cartridge.

About the NETCONF Network Discovery Cartridge

The Network Configuration (NETCONF) Network Discovery cartridge supports network discovery by using the NETCONF protocol. It generates a representation of the network infrastructure that comprises both logical and physical device hierarchies. The logical hierarchy features a logical device, child interfaces, sub-interfaces and device interface configurations. The physical hierarchy features the physical device, equipment, holders, and ports. This cartridge also creates associations between these hierarchies at the device level (between physical and logical devices) and at the interface level (between physical ports and interfaces).

Note

Before using the NETCONF Network Discovery cartridge, you must ensure that your network devices support the NETCONF protocol and Yet Another Next Generation (YANG) models. Network Integrity relies on standard NETCONF/YANG interfaces for communication and does not offer tools to activate these protocols. To enable this support, contact the device manufacturer or vendor.

About the NETCONF Protocol

The Network Configuration Protocol (NETCONF), developed and standardized by the Internet Engineering Task Force (IETF) is used to manage network device configurations. This protocol provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses XML-based encoding for both configuration data and protocol messages. NETCONF performs its operations through remote procedure calls (RPCs), which help reduce the complexity of network communication.

Inputs for NETCONF are defined by the RPC operations as listed below.

- **copy-config:** Copies one configuration datastore to another
- **delete-config:** Deletes a configuration datastore
- **get-config:** Retrieves all or part of a configuration datastore
- **get:** Retrieves all or part of an operational datastore
- **edit-config:** Changes the contents of a configuration datastore

The below examples show sample NETCONF Request and responses.

Example 1: Sample NETCONF Request to retrieve all operational data for interfaces:

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

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

Example 2: Sample NETCONF Response:

```
<rpc-reply message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabitEthernet0/0/0</name>
        <description>Main uplink interface</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
          ianaift:ethernetCsmacd
        </type>
        <enabled>true</enabled>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

About YANG Models

YANG is a data modeling language standardized by the IETF (Internet Engineering Task Force). It defines how to represent configuration data, state data, RPCs, and notifications for network management protocols. YANG is an API contract language, which defines how clients and servers interact through an API.

A YANG specification is called a YANG module, and a group of modules is often referred to as a YANG model. A YANG model serves as the schema of a network device, similar to a database schema or blueprint. This schema outlines the structure and types of data the system can exchange—such as possible configuration settings, monitoring data, notifications, and available actions. This is different from instance data, which shows the actual configuration and real-time monitoring values in use. While the YANG model defines what is possible, the instance data reflects the current state of the system.

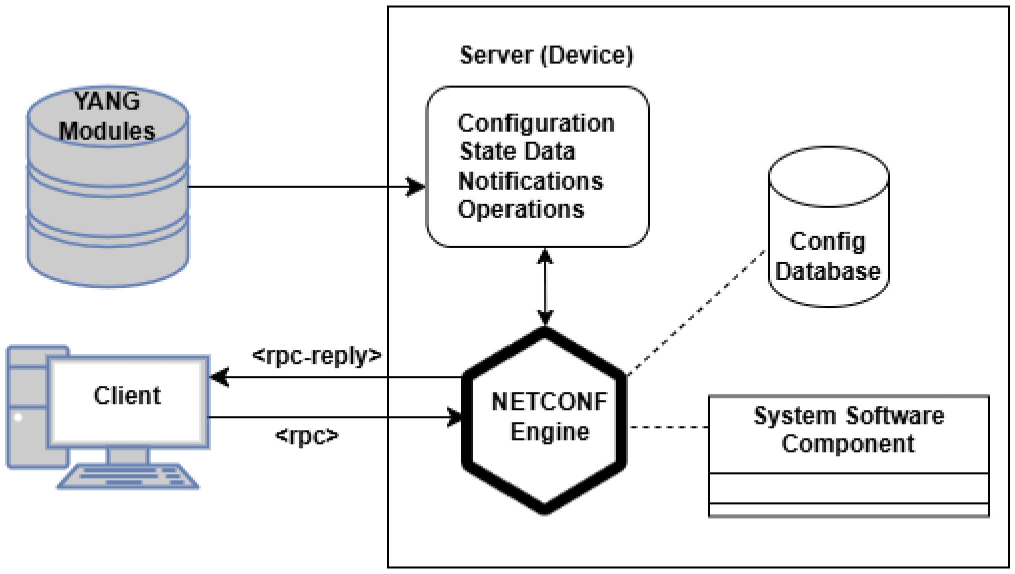
Architecture for Network Management using NETCONF and YANG Models

Figure 1-1 shows the architecture for network management using NETCONF and YANG models.

Note

In order to use YANG for network management, YANG modules must be defined to model the target domain. These modules must be then loaded, compiled, or coded into the server.

Figure 1-1 Architecture for Network Management Using NETCONF and YANG Models



The following sequence of events describe a typical client/server interaction for this architecture:

- The client application initiates a NETCONF session with the server, establishing a connection.
- The client and server exchange `<hello>` messages, containing the list of capabilities supported by each side, and learns about the modules the server supports.
- The client builds and sends an XML message with an operation in an `<rpc>` element.
- The server receives and parses the `<rpc>` element, then verifies the request against the data model defined in its YANG module.
- The server performs the operation, possibly changing the configuration datastore.
- The server builds the response, including the result, any requested data, and errors.
- The server sends XML response in an `<rpc-reply>` element.
- The client receives and parses the `<rpc-reply>` element, analyzes it, and processes it as needed.

NETCONF Communication between Network Integrity and Devices

When using NETCONF protocol to discover network devices, Network Integrity acts as a NETCONF client. This allows Network Integrity to communicate with the devices in the network and allows to retrieve device details, device status and configure the device.

When NI interacts with the devices using NETCONF protocol, it does the following:

1. The netconfConnectionInitializer processor acts as a NETCONF client and opens a NETCONF session with the device.
2. The client sends a hello message to the device:

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

```
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </capabilities>
</hello>
```

3. The device replies with a hello message that includes a list of its supported capabilities. Based on the NETCONF version in the response, the client chooses a framing mechanism:
   - NETCONF 1.0 version uses **End-of-Message Framing**.
   - NETCONF 1.1 version uses **Chunked Framing**.
4. The client builds an XML request and sends it using the selected framing method.
5. The client receives the response from the device and decodes it.

About Supported Routing Protocols

This section provides information on the different routing protocol details that the NETCONF Network Discovery and UIM Integration cartridge can discover, as listed below:

- Open Shortest Path First (OSPF)
- Link Layer Discovery Protocol (LLDP)
- Routing Information Protocol (RIP)
- Internet Protocol (IP)
- Border Gateway Protocol (BGP)
- Layer 2 Virtual Private Network (L2VPN)

Open Shortest Path First (OSPF)

OSPF is a link-state Interior Gateway Protocol (IGP) used for routing within an autonomous system. Each router running OSPF exchanges link-state advertisements (LSAs) with its neighbors and builds a complete map of the network topology.

Network Integrity uses the **ietf-ospf.yang** file (subset of the **ietf-routing.yang** model) to discover OSPF details. The attributes **dr-ip-addr**, **neighbor-router-id** and **address** discovered from this YANG model can be used to create A-end and Z-end information.

Link Layer Discovery Protocol (LLDP)

LLDP operates as a Layer 2 discovery protocol that provides a standardized method to encapsulate information about device capabilities, management address, device ID, and interface ID into LLDP frames. This protocol allows Network Integrity to discover the topology of connected network devices, and display paths between clients, switches, routers, application servers, and network servers.

Network Integrity uses the **openconfig-lldp.yang** file to discover LLDP information. The attributes, interface name and management-address, discovered from this YANG file serve to create A-end and Z-end information. The IP Assimilation cartridge uses this protocol to create topological links.

Routing Information Protocol (RIP)

RIP functions as a dynamic routing protocol based on the distance-vector algorithm. Routers using RIP exchange routing tables periodically with neighbors, using hop count as the exclusive metric for route selection. The protocol limits the maximum hop count to 15, defining the reachable network scope. Periodic updates maintain routing information, while convergence occurs over time according to the protocol's timing mechanisms. RIP remains configured primarily in small or legacy network environments.

Network Integrity uses the **ietf-rip.yang** file to discover RIP information. The attributes discovered include IPv4 and IPv6 information: address and prefix. The next-hop address contained in the YANG model is used to identify the Z-end interface address.

Internet Protocol (IP)

The Internet Protocol (IP) is the core protocol at the network layer that provides addressing and routing for communication across networks. Every device connected to an IP network is assigned a unique IP address (IPv4 or IPv6), which allows packets to be delivered from a source host to a destination host, even across multiple intermediate routers.

Network Integrity uses the **ietf-ip.yang** file (a subset of the **ietf-interfaces.yang** model) to discover IPV4 or IPV6 address, sub-netmask and prefix for the A-end Device Interface. This information is used in IP Assimilation cartridge to create A-end Device and Interface information for Links.

Border Gateway Protocol (BGP)

BGP operates as an Exterior Gateway Protocol that exchanges routing information between distinct autonomous systems. Unlike IGPs which function within a single asynchronous system, BGP uses a path-vector mechanism and associates routes with attributes such as **AS-path** to enable policy-based routing, control traffic flow, and prevent routing loops. BGP supports scalable and policy-driven route management across complex network environments.

Network Integrity discovers neighbor-address and local-address attributes from the **openconfig-bgp.yang** file, which is used for routing details.

BGP supports 2 operation types described below:

- **External BGP (eBGP):** Exchanges routing information between routers in different autonomous systems, typically at edge or border routers.
- **Internal BGP (iBGP):** Distributes routing information between routers within the same autonomous system to ensure consistent routing policies across the network.

Layer 2 Virtual Private Network (L2VPN)

L2VPN extends Layer 2 networks across wide-area backbones by enabling geographically dispersed sites to operate as part of the same Ethernet segment or LAN. It is implemented using Multiprotocol Label Switching (MPLS) pseudowires or IP-based tunneling techniques to encapsulate Layer 2 frames and transport them across the underlying infrastructure.

The solution remains transparent to Layer 3 protocols, allowing independent routing configurations within the extended Layer 2 domain. L2VPN supports network connectivity by delivering Layer 2 transport services without altering internal routing operations.

About Cartridge Dependencies

This section provides information on dependencies that the NETCONF Network Discovery cartridge has on other cartridges.

Run-Time Dependencies

This cartridge requires that the following cartridges be deployed to Network Integrity:

- Address_Handlers
- UIM_Integration_Cartridge

Design-Time Dependencies

The NETCONF Network Discovery cartridge has the following dependencies:

- Address_Handlers
- NetworkIntegritySDK
- ora_ni_uim_yang_model
- ora_uim_model
- UIM_Integration_Cartridge
- netconf-yang-model-8.0.0.jar

About the NETCONF YANG Model

The **netconf-yang-model-8.0.0.jar** is a NETCONF YANG model used in the NETCONF Network Discovery and UIM Intgeration cartridge is provided along with the cartridge. This model is generated from the YANG files used in the cartridge.

This NETCONF YANG model is generated from the Apache Maven project **yang_sources_generator**. This project can also be used to generate Java classes from YANG files of discovered devices. It is provided along with the NETCONF YANG model in the cartridge.

The project hierarchy is as shown below:

```
yangs_sources_generator
  yangs_netconf
     src/main/netconf-yang
     pom.xml
```

To generate Java classes for other YANG files:

1. Provide required YANG files (of the network devices) along with all the dependency YANG files in **yangs_netconf/src/main/netconf-yang** directory.
2. Run the command `mvn clean install` in the **yangs_netconf** directory.
   The updated **netconf-yang-model-8.0.0.jar** will be generated in the **yangs_netconf/target** directory.

Opening the Cartridge Files in Design Studio

To use the NETCONF Network Discovery cartridge, you must first download the Oracle Communications NETCONF Network Discovery Cartridge software from the Oracle software delivery web site:

https://edelivery.oracle.com

The software contains the Netconf Network Discovery cartridge ZIP file, which has the following structure:

- UIM_Cartridge_Projects\
- Network_Integrity_Cartridge_Projects
- Address_Handlers.iar
- Netconf_Network_Discovery_Cartridge.iar

For more information about opening files in Design Studio, see *Design Studio Modeling Network Integrity* and *Network Integrity Developer's Guide*.

Building and Deploying the Cartridge

See *SCD Modeling Network Integrity* for information about building and deploying cartridges.

2

About the Cartridge Components

This chapter provides information about the components of the Oracle Communications Network Integrity NETCONF Network Discovery and UIM Integration cartridge.

This cartridge contains the following actions:

- Abstract Discover Netconf Devices
- Discover Netconf Devices
- Import Netconf Device from UIM
- Detect Netconf Device Discrepancies
- Resolve Netconf Device Discrepancies

Abstract Discover Netconf Devices

The Abstract Discover Netconf Devices is an abstract action that can be extended in Oracle Communications Service Catalog and Design - Design Studio to discover specified network objects.

This action contains the following processors run in the following order:

- Netconf Connection Initializer
- Generic Netconf Yang Collector Processor
- Generic Netconf Data Discovery
- Vendor-Based Netconf Yang Collector Processor
- Vendor Netconf Data Discovery
- Netconf Logical Data Modeler
- Netconf Physical Data Modeler

Netconf Connection Initializer

The Netconf Connection Initializer processor accepts connection parameters from the **netconfParameters** scan parameter group and establishes a NETCONF connection with the device. All properties are populated into a JavaBean class named **NetconfProperties**. NETCONF Connection is established using NetconfProperties.

Generic Netconf Yang Collector Processor

The Generic NETCONF YANG Collector Processor collects variables from a device for discovery. It uses attributes from generic YANG models required to retrieve device details. This processor runs when the **deviceType** parameter is set to **Generic Device**. For more information, see About Poll Lists.

Generic Netconf Data Discovery

The Generic Netconf Data Discovery processor frames a NETCONF request to get the required attribute values from device, collects response from device and generate Java binding objects for collected response.

This processor is conditional: it is only run when the **deviceType** parameter is set to **Generic Device**.

Vendor-Based Netconf Yang Collector Processor

The Vendor-based Netconf Yang Collector Processor collects variables to be discovered from a device. This processor contains attributes from vendor-specific YANG models which are required to get device details. For more information, see About Poll Lists.

This processor is conditional: it is only run when the **deviceType** parameter is set to **Cisco XR Device**.

Vendor Netconf Data Discovery

The Vendor Netconf Data Discovery processor frames a NETCONF request to get required attribute values from the device, collects a response from the device and generates Java binding objects for the collected response.

This processor is conditional: it is only run when the **deviceType** parameter is set to **Cisco XR Device**.

Netconf Logical Data Modeler

The Netconf Logical Data Modeler processor models the data that is collected by the YANG Collector processor. Modeling includes building the hierarchical relationship of logical device and child interfaces.

Netconf Physical Data Modeler

The Netconf Physical Data Modeler processor models the data that is collected by the YANG Collector processor. Modeling includes building the hierarchical relationship of physical devices, equipment, equipment holders, and physical ports.

Discover Netconf Devices

The Discover Netconf Devices action scans devices and provides a physical and logical hierarchical model of what is discovered. This action also models the associations between the physical and logical hierarchies.

This action extends the Abstract Discover Netconf Devices action.

The Discover Netconf Devices action contains the following processors, run in the following order:

- Netconf Connection Initializer (inherited)
- Generic Netconf Yang Collector Processor (inherited)
- Generic Netconf Data Discovery (inherited)

- Vendor based Netconf Yang Collector Processor (inherited)
- Vendor Netconf Data Discovery (inherited)
- Netconf Logical Modeler (inherited)
- Netconf Physical Modeler (inherited)

Import Netconf Device from UIM

The Import Netconf Device from UIM action imports logical device and physical device trees from Oracle Communications Unified Inventory Management (UIM). Import scan parameters are available on the **Create Scan** page in Network Integrity to configure import filters.

This import action extends the Abstract Import from UIM action (from the UIM Integration cartridge) and inherits all its processors. For more information about the inherited processors in this action, see *Network Integrity UIM Integration Cartridge Guide*.

The Import Netconf Device from UIM action contains the following processors run in the following order:

1. Import UIM Initializer (inherited)
2. ProcessNetconfImportScanInput
3. Logical Device UIM Finder (inherited)
4. Physical Device UIM Finder (inherited)
5. Logical Device UIM MultiThread Importer (inherited)
6. Physical Device UIM MultiThread Importer (inherited)
7. Logical Device UIM Importer (inherited)
8. Linked Physical Device UIM Importer (inherited)
9. Logical Device UIM Persister (inherited)
10. Physical Device UIM Importer (inherited)
11. Linked Logical Device UIM Importer (inherited)
12. Physical Device UIM Persister (inherited)

ProcessNetconfImportScanInput

This processor uses filters as input parameters and sets filter values based on the scan parameters. For more information about the scan parameter groups associated with this action, see Import Scan Parameter Group.

Detect Netconf Device Discrepancies

The Detect Netconf Device Discrepancies action detects discrepancies between discovery scan results of the Discover Netconf Devices action and the data imported from UIM.

This action extends the Abstract Detect UIM Discrepancies action (from the UIM Integration cartridge) and inherits all its processors. For information about the inherited processors, see *Network Integrity UIM Integration Cartridge Guide*.

This action contains the following processors run in the following order:

1. UIM Discrepancies Filter Initializer (inherited)

2. Netconf Discrepancy Filters Initializer
3. Discrepancy Detector (inherited)

Netconf Discrepancy Filters Initializer

This processor adds the following filters to ignore association discrepancies between logical and physical devices:

- **Disregard Association+** and **Association-** discrepancies on Logical Device
- **Disregard Association+** and **Association-** discrepancies on Device Interface

Note

Discrepancy detection between logical and physical devices is required only when a different import action is used to import both logical and physical devices.

Resolve Netconf Device Discrepancies

The Resolve Netconf Device Discrepancies action resolves discrepancies on logical and physical hierarchies and associations between logical and physical entities in UIM.

This action extends the Abstract Resolve in UIM action (from the UIM Integration cartridge) and inherits all its processors. For information about the inherited processors, see *Network Integrity UIM Integration Cartridge Guide*.

The Resolve Netconf Device Discrepancies action contains the following processors run in the following order:

1. UIM Resolution Framework Initializer (inherited)
2. UIM Resolution Initializer (inherited)
3. UIM Resolution Framework Dispatcher (inherited)

3

About Poll Lists

This chapter provides a poll list for the Oracle Communications Network Integrity NETCONF Network Discovery and UIM Integration cartridge YANG processor that collects details of variables to be discovered from devices.

About the Generic Netconf Yang Collector Processor Poll List

The Generic Netconf Yang Collector Processor poll list id:

- **ietf-system.yang**

  ```
  /ietf-system:system/hostname
  /ietf-system:system/location
  ```

- **ietf-interfaces.yang**

  ```
  /ietf-interfaces:interfaces/interface/name
  /ietf-interfaces:interfaces/interface/description
  /ietf-interfaces:interfaces/interface/type
  /ietf-interfaces:interfaces/interface/enabled
  /ietf-interfaces:interfaces-state/interface/name
  /ietf-interfaces:interfaces-state/interface/type
  /ietf-interfaces:interfaces-state/interface/admin-status
  /ietf-interfaces:interfaces-state/interface/oper-status
  /ietf-interfaces:interfaces-state/interface/last-change
  /ietf-interfaces:interfaces-state/interface/if-index
  /ietf-interfaces:interfaces-state/interface/phys-address
  /ietf-interfaces:interfaces-state/interface/speed
  ```

- **ietf-ip.yang**

  ```
  /ietf-interfaces:interfaces/interface/ietf-ip:ipv4/address/ip
  /ietf-interfaces:interfaces/interface/ietf-ip:ipv4/address/subnet
  /ietf-interfaces:interfaces/interface/ietf-ip:ipv6/address/ip
  /ietf-interfaces:interfaces/interface/ietf-ip:ipv6/address/prefix-length
  ```

- **ietf-routing.yang**

  ```
  /ietf-routing:routing/control-plane-protocols/control-plane-protocol/name
  /ietf-routing:routing/control-plane-protocols/control-plane-protocol/type
  ```

- **ietf-ospf.yang**

  ```
  /ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
  ospf:ospf/areas/area/area-id
  /ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
  ospf:ospf/areas/area/interfaces/interface/name
  /ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
  ```

```
ospf:ospf/areas/area/interfaces/interface/interface-type
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
ospf:ospf/areas/area/interfaces/interface/dr-ip-addr
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
ospf:ospf/areas/area/interfaces/interface/state
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
ospf:ospf/areas/area/interfaces/interface/neighbors/neighbor/neighbor-
router-id
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
ospf:ospf/areas/area/interfaces/interface/neighbors/neighbor/address
```

- **ietf-rip.yang**

```
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv4/routes/route/ipv4-prefix
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv4/routes/route/next-hop
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv4/routes/route/interface
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv6/routes/route/ipv4-prefix
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv4/routes/route/next-hop
/ietf-routing:routing/control-plane-protocols/control-plane-protocol/ietf-
rip:rip/ipv6/routes/route/interface
```

- **openconfig-bgp.yang**

```
/bgp:bgp/neighbors/neighbor/neighbor-address
/bgp:bgp/peer-groups/peer-group/transport/config/local-address
```

- **openconfig-lldp.yang**

```
/lldp:lldp/interfaces/interface/name
/lldp:lldp/interfaces/interface/neighbors/neighbor/id
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/system-name
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/system-description
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/port-id
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/port-id-type
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/port-description
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/management-address
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/system-name
/lldp:lldp/interfaces/interface/neighbors/neighbor/state/system-description
```

- **ietf-l2vpn-ntw.yang**

```
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-description
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/vpn-
node-id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/global-parameters-
profiles/global-parameters-profile/profile-id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/bgp-
auto-discovery/rd-choice
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/bgp-
```

```
auto-discovery/vpn-target/route-targets/route-target
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/
active-global-parameters-profiles/global-parameters-profile/profile-id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/vpn-
network-accesses/vpn-network-access/id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/vpn-
network-accesses/vpn-network-access/interface-id
/ietf-l2vpn-ntw:l2vpn-ntw/vpn-services/vpn-service/vpn-nodes/vpn-node/bgp-
auto-discovery/vpn-target/id
```

- **ietf-hardware.yang**

```
/ietf-hardware:hardware/component/name
/ietf-hardware:hardware/component/is-fru
/ietf-hardware:hardware/component/class
/ietf-hardware:hardware/component/physical-index
/ietf-hardware:hardware/component/description
/ietf-hardware:hardware/component/parent
/ietf-hardware:hardware/component/contains-child
/ietf-hardware:hardware/component/hardware-rev
/ietf-hardware:hardware/component/software-rev
/ietf-hardware:hardware/component/serial-num
/ietf-hardware:hardware/component/mfg-name
/ietf-hardware:hardware/component/model-name
/ietf-hardware:hardware/component/alias
```

About the Vendor-Based Netconf Yang Collector Processor Poll List

The Vendor-based Netconf Yang Collector Processor poll list is:

- **Cisco-IOS-XR-snmp-agent-oper.yang**

```
/Cisco-IOS-XR-snmp-agent-oper:snmp/information/system-name/system-name
/Cisco-IOS-XR-snmp-agent-oper:snmp/information/system-descr/sys-descr
/Cisco-IOS-XR-snmp-agent-oper:snmp/information/system-oid/sys-obj-id
```

- **Cisco-IOS-XR-pfi-im-cmd-oper.yang**

```
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/interface-
name
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/mac-address
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/state
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/line-state
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/interface-
type
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/mtu
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/speed
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/bandwidth
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/max-
bandwidth
/Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/parent-
interface-name
```

- **Cisco-IOS-XR-ifmgr-cfg.yang**
  `/Cisco-IOS-XR-ifmgr-cfg:interface-configurations/interface-configuration/interface-name`
- **Cisco-IOS-XR-ipv4-io-cfg.yang**

  ```
  /Cisco-IOS-XR-ifmgr-cfg:interface-configurations/interface-configuration/
  Cisco-IOS-XR-ipv4-io-cfg:ipv4-network/addresses/primary
  /Cisco-IOS-XR-ifmgr-cfg:interface-configurations/interface-configuration/
  Cisco-IOS-XR-ipv4-io-cfg:ipv4-network/addresses/secondaries
  ```

- **Cisco-IOS-XR-ipv6-ma-cfg.yang**
  `/Cisco-IOS-XR-ifmgr-cfg:interface-configurations/interface-configuration/Cisco-IOS-XR-ipv6-ma-cfg:ipv6-network/addresses/regular-addresses`
- **Cisco-IOS-XR-ethernet-lldp-oper.yang**

  ```
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/global-lldp/lldp-info/system-name
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/summaries/
  summary/lldp-neighbor/detail/port-description
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/summaries/
  summary/lldp-neighbor/detail/system-name
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/summaries/
  summary/lldp-neighbor/detail/system-description
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/summaries/
  summary/lldp-neighbor/mib/port-id-sub-type
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/summaries/
  summary/lldp-neighbor/port-id-detail
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface/
  interface-name
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface/
  local-network-addresses/lldp-addr-entry/address
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface/port-
  id
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface/port-
  id-sub-type
  /Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface/port-
  description
  ```

- **Cisco-IOS-XR-ip-rip-oper.yang**

  ```
  /Cisco-IOS-XR-ip-rip-oper:rip/default-vrf/interfaces/interface/interface-
  name
  /Cisco-IOS-XR-ip-rip-oper:rip/default-vrf/interfaces/interface/destination-
  address
  /Cisco-IOS-XR-ip-rip-oper:rip/default-vrf/interfaces/interface/rip-summary/
  next-hop-address
  /Cisco-IOS-XR-ip-rip-oper:rip/default-vrf/interfaces/interface/rip-summary/
  prefix
  ```

- **Cisco-IOS-XR-ipv4-bgp-oper.yang**

  ```
  /Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-standby/
  default-vrf/afs/af/neighbor-af-table/neighbor/neighbor-address
  /Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-standby/
  attributes/attribute/attribute-info/ribrnh-ip/ipv4-address
  ```

```
/Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-standby/
attributes/attribute/attribute-info/ribrnh-ip/ipv6-address
/Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-standby/
default-vrf/afs/af/neighbor-af-table/neighbor/connection-state
```

- **Cisco-IOS-XR-ipv4-ospf-oper.yang**

```
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/process-name
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
area-id
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
neighbors/neighbor/interface-name
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
neighbors/neighbor/neighbor-address
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
interface-briefs/interface-brief/interface-name
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
interface-briefs/interface-brief/interface-address
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/areas/area/
interface-briefs/interface-brief/ospf-interface-state
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/process-
information/process-areas/process-area/area-id
/Cisco-IOS-XR-ipv4-ospf-oper:ospf/processes/process/default-vrf/process-
information/process-areas/process-area/area-lsa-count
```

- **Cisco-IOS-XR-mpls-vpn-oper.yang**

```
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/vrf-name
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/vrf-description
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/route-distinguisher
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/interface/interface-name
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/af/route-target/route-target-
type
/Cisco-IOS-XR-mpls-vpn-oper:l3vpn/vrfs/vrf/af/route-target/route-target-
value
```

- **Cisco-IOS-XR-invmgr-oper.yang**

```
/Cisco-IOS-XR-invmgr-oper:inventory/racks/rack/name
/Cisco-IOS-XR-invmgr-oper:inventory/racks/rack/attributes/inv-basic-bag
/Cisco-IOS-XR-invmgr-oper:inventory/racks/rack/entity
```

4

Using the Cartridge

This chapter explains how to use the Oracle Communications Network Integrity NETCONF Network Discovery and UIM Integration cartridge.

Creating a Discover Netconf Devices Scan

The NETCONF Network Discovery cartridge allows you to create a Discover Netconf Devices scan.

To create a Discover Netconf Devices scan:

1. Create a new scan.

   See *Network Integrity Online Help* for more information about creating a new scan.
2. On the **General** tab, select **Discover Netconf Devices** from the Scan Action list.

   The **Scan Type** field displays **Discovery**.
3. In the **Scan Action Parameters** section, configure the scan with the following parameters.

   Enter the following parameters:
   - In the **Username** field, enter the username to establish connection with device.
   - In the **Password** field, enter the password to establish connection with device.
   - In the **Port** field, enter on which port of the device Netconf is running.
   - In the **deviceType** field, select the type of device to be discovered.
   - In the **NetconfVersion** field, select the Netconf version supported by device.
   - In the **ConnectionTimeOut** field, provide maximum time limit in milliseconds to establish connection.
   - In the **CommandTimeOut** field, provide maximum time limit in milliseconds for command execution.
   - In the **subSystem** field, specify the SSH subsystem to use.
   - In the **serverHostKey** field, provide the expected host key algorithm for the SSH server.
4. Provide device IP Address in scope.
5. Save and run the scan.
   The scan discovers and models logical and physical devices. It also creates a device model for each logical and physical device.

Populating UIM with Discovered Data

The NETCONF Network Discovery and UIM Integration cartridge allows you to populate Oracle Communications Unified Inventory Management (UIM) with network data discovered by the Discover Netconf Devices discovery action.

To populate UIM with discovered network data:

1. Create a new scan.

   See *Network Integrity Online Help* for more information about creating a new scan.
2. On the **General** tab of the **Create Scan** page:
   a. From the Scan Action list, select **Discover Netconf Devices**.
      The **Scan Type** field displays **Discovery**.
   b. Select **Detect Discrepancies**.
   c. In the **Scan Action Parameters** area, provide any necessary information.
3. Provide device IP Address in scope.
4. Run the discovery scan.
   The scan generates **Entity+** discrepancies for each discovered device.
5. Right-click on the discrepancies you want to populate into UIM and select **Correct Netconf Device Discrepancies in UIM**.
6. Click **Submit**.

Import Reconciled Data from UIM

The NETCONF Network Discovery and UIM Integration cartridge allows you to import reconciled data from UIM into Network Integrity.

To create an Import Netconf Device from UIM scan:

1. Create a new scan.

   See *Network Integrity Online Help* for more information.
2. On the **General** tab of the **Create Scan** page, do the following:
   a. From the Scan Action list, select **Import Netconf Device from UIM**.
      The **Scan Type** field displays **Import**.
   b. In the **Scan Action Parameters**, add any filters if required.
3. Provide any necessary information.
4. Save and run the scan.

   The logical and physical device models are created for each device.

5 About Cartridge Modeling

The Oracle Communications Network Integrity NETCONF Network Discovery and UIM Integration cartridge models collected data according to the Oracle Communications Information Model. Collected data is modeled into the following entities:

- DeviceInterfaceConfiguration
- DeviceInterfaceConfigurationItem
- Equipment
- EquipmentHolder
- EquipmentEquipmentRel
- EquipmentHolderEquipmentRel
- LogicalDevice
- MediaInterface
- PhysicalDevice
- PhysicalDeviceEquipmentRel
- PhysicalPort

See the following topics for more information about modeling cartridges:

- Hierarchy Mapping
- Oracle Communications Information Model Information
- Specifications
- Specification Cardinality
- Equipment Visual Specifications
- Field Mapping

Hierarchy Mapping

This section describes the hierarchy mapping for the NETCONF Network Discovery cartridge.

Generic Device

The data sourced from the **ietf-system.yang** file establishes and seeds the logical and physical device objects.

The media interface encapsulates the common information about an interface as a device is discovered. The device interface configuration captures the media type information that decorates the interface with media-specific parameters. These media-specific parameters define the behavior of the interface.

The media interfaces are established and seeded with data sourced from the following YANG files:

- **ietf-interfaces.yang**
- **ietf-ip.yang**
- **ietf-routing.yang**
- **ietf-ospf.yang**
- **ietf-rip.yang**
- **openconfig-lldp.yang**
- **openconfig-bgp.yang**
- **ietf-l2vpn-ntw.yang**

Equiments, Equiment Holders and Physical Ports are established and seeded with data sourced from the following YANG files:

- **ietf-hardware.yang**

Vendor-Based Device

The data sourced from the **Cisco-XR-snmp-agent-oper.yang** file establishes and seeds the logical and physical device object.

The media interface encapsulates the common information about an interface as a device is discovered. The device interface configuration captures the media type information that decorates the interface with media-specific parameters. These media-specific parameters define the behavior of the interface.

The media interfaces are established and seeded with data sourced from the following YANG files:

- **Cisco-IOS-XR-pfi-im-cmd-oper.yang**
- **Cisco-IOS-XR-ifmgr-cfg.yang**
- **Cisco-IOS-XR-ethernet-lldp-oper.yang**
- **Cisco-IOS-XR-ip-rip-oper.yang**
- **Cisco-IOS-XR-ipv4-bgp-oper.yang**
- **Cisco-IOS-XR-ipv4-io-cfg.yang**
- **Cisco-IOS-XR-ipv4-ospf-oper.yang**
- **Cisco-IOS-XR-ipv6-ma-cfg.yang**
- **Cisco-IOS-XR-mpls-vpn-oper.yang**

Equiments, Equiment Holders and Physical Ports are established and seeded with data sourced from the following YANG files:

- **Cisco-IOS-XR-invmgr-oper.yang**

Oracle Communications Information Model Information

All entities used in NETCONF Network Discovery and UIM Integration cartridge (for example, physical device, logical device, media interface, and so on) are Oracle Communications Information Model 1.0-compliant for static fields. The dynamic fields (sometimes referred to as characteristics) are application-specific. You can customize application specific data with the device interface configuration mechanism.

The NETCONF Network Discovery Cartridge supports the **Generic Media** configuration.

Specifications

This section lists the specifications included in the ora_ni_uim_yang_model cartridge project for modeling devices.

You must first model inventory specifications in an inventory cartridge using Design Studio, define the cartridge dependency such that the Network Integrity cartridge is dependent on the inventory cartridge, and then use the inventory cartridge specifications in the Network Integrity cartridge model.

Specifications shared with UIM are defined in the ora_ni_uim_yang_model cartridge project. These projects are used to directly deploy specifications to UIM.

Logical Device

Table 5-1 shows the list of logical device specifications.

Table 5-1 Logical Device Specifications

Specification	Cartridge	Intended Usage
logicalDeviceSpecification	ora_ni_uim_yang_model	Used to model all types of devices.

Table 5-2 shows the characteristics applied to the logical device specifications.

Table 5-2 Logical Device Characteristics

Characteristics	Field Type	Field Content
deviceIPAddress	String	Text
nativeEmsName	String	Text

Device Interface

Device Interface specifications are set based on speed of interface.

Table 5-3 shows the list of device interface specifications.

Table 5-3 Device Interface Specifications

Specification	Cartridge	Intended Usage
deviceInterfaceSpecification	ora_ni_uim_yang_model	Used to model device interfaces with undefined ratecode.
400GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 400GigE ratecode.
200GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 200GigE ratecode.
100GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 100GigE ratecode.
50GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 50GigE ratecode.

Table 5-3 (Cont.) Device Interface Specifications

Specification	Cartridge	Intended Usage
25GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 25GigE ratecode.
15GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 15GigE ratecode.
10GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 10GigE ratecode.
5GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 5GigE ratecode.
1GigE_interface	ora_ni_uim_yang_model	Used to model device interfaces with 1GigE ratecode.
100M_interface	ora_ni_uim_yang_model	Used to model device interfaces with 100M ratecode.
10M_interface	ora_ni_uim_yang_model	Used to model device interfaces with 10M ratecode.
subInterfaceSpecification	ora_ni_uim_yang_model	Used to model all sub interfaces.

Table 5-4 shows the characteristics applied to deviceInterfaceSpecification specification.

Table 5-4 Device Interface Item Characteristics for deviceInterfaceSpecification Specification

Characteristics	Field Type	Field Content	Description
highSpeed	String	Text	High Speed supported on device
ifSpeed	String	Text	Speed
nativeEMSAdminServiceState	String	Text	Admin Service State
nativeEMSServiceState	String	Text	Service State
nativeEmsName	String	Text	A unique identifier such as name
bgpPeerAdminStatus	String	Text	Administrative state of the BGP peer
bgpPeerLastError	String	Text	Last error recorded with the BGP peer
bgpPeerLocalAddr	String	Text	Local IP address used in the BGP session
bgpPeerRemoteAddr	String	Text	Remote IP address of the BGP peer
ipCidrRouteDest	String	Text	Destination address for a route
ipCidrRouteMask	String	Text	Subnet mask associated with a route
ipCidrRouteNextHop	String	Text	Next-hop IP address for a route
AEndDeviceName	String	Text	Name of device on A end
AEndInterfaceIPAddress	String	Text	A end interface IP Address

Table 5-4 (Cont.) Device Interface Item Characteristics for deviceInterfaceSpecification Specification

Characteristics	Field Type	Field Content	Description
AEndInterfaceName	String	Text	Name of interface on A end
ZEndDeviceName	String	Text	Name of device on Z end
ZEndInterfaceIPAddress	String	Text	Z end interface IP Address
ZEndInterfaceName	String	Text	Name of interface on Z end
mplsVpnVrfDescription	String	Text	Description of the VPN Routing and Forwarding (VRF)
mplsVpnVrfRouteDistinguisher	String	Text	Route distinguisher for identifying unique VPNs
rtBothValue	String	Text	Route targets for both import and export
rtExportValue	String	Text	Route target(s) used for route export
rtImportValue	String	Text	Route target(s) used for route import
ospfIfAdminState	String	Text	Administrative state of the OSPF interface
ospfIfAreaId	String	Text	OSPF area identifier for the interface
ospfIfIpAddress	String	Text	IP address associated with the OSPF interface
ospfIfLsaCount	String	Text	Count of LSAs seen or generated on interface
ospfIfState	String	Text	Operational state of OSPF (e.g., Down, DR, BDR)
ospfIfType	String	Text	Interface type (broadcast, point-to-point, etc.)
ospfNbrIpAddr	String	Text	IP address of the adjacent OSPF neighbor

Device Interface Configuration Item

Table 5-5 shows the list of Device Interface specifications.

Table 5-5 Device Interface Configuration Item Specifications

Specification	Cartridge	Intended Usage
Generic_DI_Config_Specification	NetworkIntegritySDK	Used to model Device Interface IPAdress Details.

shows the characteristics applied to Generic_IPAddress Device Interface Configuration Items.

Table 5-6 Device Interface Configuration Item Characteristics for Generic_DI_Config_Specification

Characteristics	Field Type	Field Content
GenericIPAddress	String	Text
GenericPrefix	String	Text
GenericIpVersion	String	Text **Note:** The allowed values are "IPV4" or "IPV6".

Physical Device

Table 5-7 shows the list of physical device specifications.

Table 5-7 Physical Device Specifications

Specification	Cartridge	Intended Usage
physicalDeviceSpecification	ora_ni_uim_yang_model	Used to model all types of devices.

Table 5-8 shows the characteristics applied to the physical device specifications.

Table 5-8 Physical Device Characteristics

Characteristics	Field Type	Field Content
deviceIPAddress	String	Text
nativeEmsName	String	Text

Equipment

Table 5-9 shows the list of Equipment specifications.

Table 5-9 Equipment Specifications

Specification	Cartridge	Intended Usage
shelfSpecification	ora_ni_uim_yang_model	Used to model all types of devices.
cardSpecification	ora_ni_uim_yang_model	Used to model all types of devices.

Table 5-10 shows the characteristics applied to the equipment specifications.

Table 5-10 Equipment Characteristics

Characteristics	Field Type	Field Content
hardwareRevision	String	Text
softwareRevision	String	Text
modelName	String	Text

Table 5-10 (Cont.) Equipment Characteristics

Characteristics	Field Type	Field Content
nativeEmsName	String	Text
vendorName	String	Text

Equipment Holder

Table 5-11 shows the list of equipment holder specifications.

Table 5-11 Equipment Holder Specifications

Specification	Cartridge	Intended Usage
holderSpecification	ora_ni_uim_yang_model	Used to model all types of devices.

Table 5-12 shows the characteristics applied to the equipment holder specifications.

Table 5-12 Equipment Holder Characteristics

Characteristics	Field Type	Field Content
nativeEmsName	String	Text

Physical Port

Table 5-13 shows the list of Physical Port specifications.

Table 5-13 Physical Port Specifications

Specification	Cartridge	Intended Usage
portSpecification	ora_ni_uim_yang_model	Used to model physical ports with no device interface associated or mapped with generic device interface.
400GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 400GigE device interface.
200GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 200GigE device interface.
100GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 100GigE device interface.
50GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 50GigE device interface.
25GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 25GigE device interface.

Table 5-13 (Cont.) Physical Port Specifications

Specification	Cartridge	Intended Usage
15GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 15GigE device interface.
10GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 10GigE device interface.
5GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 5GigE device interface.
1GigE_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 1GigE device interface.
100M_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 100M device interface.
10M_Port	ora_ni_uim_yang_model	Used to model physical ports mapped with 10M device interface.

Table 5-13 shows the characteristics applied to the physical port specifications.

Table 5-14 Physical Port Characteristics

Characteristics	Field Type	Field Content
nativeEmsName	String	Text

Specification Cardinality

The cardinality of all specification parent-child relationships is `min=0` and `max=n`. This approach allows Network Integrity to programmatically instantiate all objects on demand as they are discovered using the web service.

Equipment Visual Specifications

The visual facility on the Equipment specifications is not used. It is left to the you to decide if you want to enrich the technology pack to provide visual effects to the UIM GUI for a given Equipment entity.

Field Mapping

The NETCONF Network Discovery Cartridge supports the following field mappings:

- **Text:** Implies Text [255]
- **static:** Information Model 1.0 defines this field to be static on the entity specification. The specification provides getters and setters for this field.

- **dynamic:** This is a dynamic field where the entity specification treats the field as a name and value pair. The specification does not provide getter and setters but generically has get and set characteristics method holding a HashSet of entries.

6 About Design Studio Construction

This chapter provides information on the composition of the Oracle Communications Network Integrity NETCONF Network Discovery Cartridge from the Oracle Communications Service Catalog and Design - Design Studio perspective.

Model Collections

The table below lists the model collection used in the NETCONF Network Discovery and UIM Integration cartridge.

Table 6-1 NETCONF Network Discovery Cartridge Model Collection

Specification	Information Model Entity Type	Intended Usage/Notes
logicalDeviceSpecification	LogicalDevice	Represents root object discovered on the network.
deviceInterfaceSpecification	DeviceInterface	Represents generic interfaces.
400GigE_interface	DeviceInterface	Represents interfaces with 400GigE rate code.
200GigE_interface	DeviceInterface	Represents interfaces with 200GigE rate code.
100GigE_interface	DeviceInterface	Represents interfaces with 100GigE rate code.
50GigE_interface	DeviceInterface	Represents interfaces with 50GigE rate code.
25GigE_interface	DeviceInterface	Represents interfaces with 25GigE rate code.
15GigE_interface	DeviceInterface	Represents interfaces with 15GigE rate code.
10GigE_interface	DeviceInterface	Represents interfaces with 10GigE rate code.
1GigE_interface	DeviceInterface	Represents interfaces with 1GigE rate code.
100M_interface	DeviceInterface	Represents interfaces with 100M rate code.
10M_interface	DeviceInterface	Represents interfaces with 10M rate code.
subInterfaceSpecification	DeviceInterface	Represents Sub Interfaces.
Generic_DI_Config_Specification	DeviceInterfaceConfigurationItem	Represents Generic Media.
physicalDeviceSpecification	PhysicalDevice	Represents any Physical Device discovered on the network.
holderSpecification	EquipmentHolder	Represents Board piece of equipment.
shelfSpecification	Equipment	Represents Shelf piece of equipment.
cardSpecification	Equipment	Represents Module piece of equipment.

Table 6-1 (Cont.) NETCONF Network Discovery Cartridge Model Collection

Specification	Information Model Entity Type	Intended Usage/Notes
portSpecification	PhysicalPort	Represents generic physical port.
400GigE_Port	PhysicalPort	Represents ports associated with 400GigE rate code.
200GigE_Port	PhysicalPort	Represents ports associated with 200GigE rate code.
100GigE_Port	PhysicalPort	Represents ports associated with 100GigE rate code.
50GigE_Port	PhysicalPort	Represents ports associated with 50GigE rate code.
25GigE_Port	PhysicalPort	Represents ports associated with 25GigE rate code.
15GigE_Port	PhysicalPort	Represents ports associated with 15GigE rate code.
10GigE_Port	PhysicalPort	Represents ports associated with 10GigE rate code.
1GigE_Port	PhysicalPort	Represents ports associated with 1GigE rate code.
100M_Port	PhysicalPort	Represents ports associated with 100M rate code.
10M_Port	PhysicalPort	Represents ports associated with 10M rate code.

Specification Lineage

This section shows examples of the logical and physical specification lineages that the NETCONF Network Discovery cartridge generates for the discovered devices in the network.

Logical Specification Lineage for Devices

The example below shows the logical specification lineage for discovered devices. This lineage shows the intended relationship between specifications.

Example Logical Specification Lineage:

```
logicalDeviceSpecification
     [0..*] deviceInterfaceSpecification
           [0..1] Generic_DI_Config_Specification
                 [0..1] Generic_IPAddresses
                       [0..*] Generic_IPAddress

                              GenericIPAddress (characteristic)
                              GenericPrefix (characteristic)
                              GenericIPVersion (characteristic)
```

Physical Specification Lineage for Devices

The example below shows the physical specification lineage for discovered devices. This lineage shows the intended relationship between specifications.

Example Physical Specification Lineage:

```
physicalDeviceSpecification
     shelfSpecification
           [0..*] holderSpecification
                   [0..1]  cardSpecification
                           [0..*] holderSpecification
                                    [0..1]  cardSpecification
                                               [0..1]  portSpecification
```

Discovery Action

The NETCONF Network Discovery Cartridge supports the discovery action Discover Netconf Devices which is used to discovery devices in the network.

Table 6-2 Discover Netconf Devices

Result Category	Address Handler	Scan Parameters	Model	Processors
Device	IPAddressHandler	netconfParameters • username • password • port • deviceType • netconfVersion • connectionTimeout • commandTimeout • subsystem • serverHostKey	Netconf_Network_Discovery_Cartridge	• Netconf Connection Intializer • Generic Netconf Yang Collector Processor • Generic Netconf Data Discovery • Vendor Based Netconf Yang Collector Processor • Vendor Netconf Data Discovery • Netconf Logical Data Modeler • Netconf Physical Data Modeler

Discovery Scan Parameter Group

The Discover Netconf Devices action uses the netconfParameter scan parameter group. The below table outlines the Design Studio construction of this scan parameter group.

Table 6-3 netconfParameters Scan Parameter Group Design Studio Construction

Characteristic name	Parameter Type	Description	UI Label
username	String	Username to establish connection	username

Table 6-3 (Cont.) netconfParameters Scan Parameter Group Design Studio Construction

Characteristic name	Parameter Type	Description	UI Label
password	String	Password to establish connection	password
port	String	Port to establish connection	port
connectionTimeout	String	Maximum time limit in milliseconds to establish connection.	connectionTimeout
commandTimeout	String	Maximum time limit in milliseconds for command execution.	commandTimeout
deviceType	Dropdown	Type of device to be discovered	deviceType
netconfVersion	Dropdown	NETCONF version supported on device	Netconf Version
subsystem	String	Specifies the SSH subsystem to use.	Subsystem
serverHostKey	String	Specifies the expected host key algorithm for the SSH server	Server Host Key
hostKeysFileName	String	Path to SSH client's Known Hosts file	hostKeysFileName
keyBasedAuthentication	CheckBox	Enable key-based authentication	keyBasedAuthentication
pemKeyFile	String	Path to SSH client's Private Key	pemKeyFile
strictHostKeyChecking	CheckBox	Enable verification of SSH server's host key.	strictHostKeyChecking

Discovery Processors

The below table describes the discovery processors of the Discover Netconf Devices action.

Table 6-4 List of Discovery Processors

Processor Name	Variable
Netconf Connection Intializer	Input: N/A Output: • deviceType • netconfConnection
Generic Netconf Yang Collector Processor	Input: directory path to YANG files. Output: • genericYangProcessorProcessorResponseType A collection of attributes to be discovered from device

Table 6-4 (Cont.) List of Discovery Processors

Processor Name	Variable
Generic Netconf Data Discovery	Input: • genericYangProcessorProcessorResponseType • netconfConnection Output: • genericbindings Map of response Java objects
Vendor Based Netconf Yang Collector Processor	Input: directory path to YANG files. Output: • vendorbasednetconfyangcollectorProcessorResponseType A collection of attributes to be discovered from device
Vendor Netconf Data Discovery	Input: • vendorbasednetconfyangcollectorProcessorResponseType • netconfConnection Output: • vendorBindings Map of response Java objects
Netconf Logical Data Modeler	Input: • genericbindings • vendorBindings Output: • logicalDevice
Netconf Physical Data Modeler	Input: • genericbindings • vendorBindings Output: • physicalDevice

Import Action

The Import Netconf Device from UIM action is used to import devices from UIM.

Table 6-5 Import Netconf Device from UIM

Result Category	Scan Parameters	Model	Processors
Device	UIMImportParameters	Netconf_Network_Discovery_Cartridge	• ProcessNetconfImportScanInput • This action extends the Abstract Import from UIM action. See *Network Integrity UIM Integration Cartridge Guide* for information about the processors in this action.

Import Scan Parameter Group

The Import from UIM action uses the UIMImportParameters scan parameter group. The below table outlines the Design Studio construction of this scan parameter group.

Table 6-6 UIMImportParameters Scan Parameter Group Design Studio Construction

Characteristic Name	Parameter Type	Description	UI Label
adminState	Dropdown	The status of the device in the inventory system.	Inventory State
importLogicalDevices	Check box	Use this box to indicate whether to import logical devices. By default, this box is checked in the UI.	Import Logical Devices
importPhysicalDevices	Check box	Use this box to indicate whether to import physical devices. By default, this box is checked in the UI.	Import Physical Devices
logicalDeviceSpecification	String	The specification name(s) for logical devices. This field supports wildcard characters. Values are comma-separated if multiple specifications are given.	Logical Device Specification
name	String	Use to filter imported devices by device name. This field supports wildcard characters.	Name
networkLocationEntityCode	String	The network or entity location code. This field supports wildcard characters.	Network/Entity Location
physicalDeviceSpecification	String	The specification name(s) for physical devices. This field supports wildcard characters. Values are comma-separated if multiple specifications are given.	Physical Device Specification

Discrepancy Detection Action

The Detect Netconf Device Discrepancies action is used to perform discrepancy detection.

Table 6-7 Detect Netconf Device Discrepancies

Result Category	Result Source	Scan Parameters	Model	Processors
All	Discover Netconf Devices	N/A	Netconf_Network_Discovery_Cartridge	Netconf Discrepancy Filters Initializer This action extends the Abstract Detect UIM Discrepancies action included in the UIM Integration cartridge. For more information, see *Network Integrity UIM Integration Cartridge Guide*.

Detect Netconf Device Discrepancies

The Detect Netconf Device Discrepancies action detects discrepancies between discovery scan results of the Discover Netconf Devices action and the data imported from UIM.

This action extends the Abstract Detect UIM Discrepancies action (from the UIM Integration cartridge) and inherits all its processors. For information about the inherited processors, see *Network Integrity UIM Integration Cartridge Guide*.

This action contains the following processors run in the following order:

1. UIM Discrepancies Filter Initializer (inherited)
2. Netconf Discrepancy Filters Initializer
3. Discrepancy Detector (inherited)

Discrepancy Resolution Action

The Reconcile Netconf Device Discrepancies action is used to perform discrepancy resolution.

Table 6-8 Reconcile Netconf Device Discrepancies

Result Category	Result Source	Processors
All	Discover Netconf Devices	This action extends the Abstract Resolve in UIM action included in the Network Integrity UIM Integration cartridge. For more information, see *Network Integrity UIM Integration Cartridge Guide*.

7 About Design Studio Extension

This chapter provides scenarios for the extensibility of NETCONF Network Discovery and UIM Integration cartridge using Oracle Communications Service Catalog and Design - Design Studio.

Adding a New Device Type under Generic

To discover and model a new device type:

- Collect all the YANG files supported by the device and place them in the YANG directory path set in Design Studio preferences.
- Generate **netconf-yang-model-8.0.0.jar** with new YANG models using yang_sources_generator project and update jar in Netconf_Network_Discovery_Cartridge/lib folder.

  For more information about generating netconf-yang-model, see About the NETCONF YANG Model.
- Create a new processor of the type **yangprocessor**. Select all attributes that you want to be discovered from device.
- Create a new **discoveryprocessor** which accepts **yangprocessor** output as input. Refer to the **VendorNetconfDataDiscoveryProcessorImpl** class to collect response from device and generate java objects.
- Update the Netconf Logical Device Modeler and Netconf Physical Device Modeler implementations to support modeling of the new device.

For more information regarding extensibility of cartridges, see *Network Integrity Developer's Guide*.

Fixing Known Issues in the NETCONF Network Discovery Cartridge

This section provides information about known issues in the NETCONF Network Discovery and UIM Integration cartridge.

Issues Occurring During Cartridge Runtime

This topic describes the issues occurring during the cartridge runtime and their fixes, as listed:

- Generation of Duplicate Class Names in a YANG File
- YANG Files with Deviation Statements May Not Be Processed

Generation of Duplicate Class Names in a YANG File

During cartridge runtime, the binding generator may generate classes with the same name in one YANG file.

Example:

The generator creates the same Java class for two similar containers in a YANG file: **Cisco-ios-xe-native:mac-address-table/learning** and **Cisco-ios-xe-native:mac/address-table/learning**. This happens because the binding generator treats both "-" and "/" in the paths as valid for creating packages, which results in the same class being genrated for both:

```
org.opendaylight.yang.gen.v1.http.cisco.com.ns.yang.cisco.ios.xe._native.rev180522._nativ
e.mac.address.table.Learning
```

The duplication causes a conflict during creation of the runtime context and generates the following error:

```
Caused by: com.google.common.base.VerifyException: Conflict on runtime type
mapping of
org.opendaylight.yang.gen.v1.http.cisco.com.ns.yang.cisco.ios.xe._native.rev18
0522._native.mac.address.table.Learning
```

Fix: To fix this issue, delete any one of the similar containers in the YANG file and re-generate the bindings JAR file.

To apply the fix for the above described example.

1. Identify the statement in the error that appears. The Java classes generated will be the same for both the statements. In this example, the statements are *statement=EmptyContainerEffectiveStatement{argument=(http://cisco.com/ns/yang/Cisco-IOS-XE-native?revision=2018-05-22)learning}}* and *statement=EmptyContainerEffectiveStatement{argument=(http://cisco.com/ns/yang/Cisco-IOS-XE-native?revision=2018-05-22)learning}}*
2. Go to the reported module where the error originates, which is **Cisco-IOS-XE-native** and find statements with "learning" in them.
3. Examine the absolute path of each node that contain "learning" and delete containers with matching paths so that only one of the matching nodes remains.
4. Re-generate the binding JAR files.
5. Use the regenerated JAR file in the NETCONF Network Discovery and UIM Integration cartridge.

YANG Files with Deviation Statements May Not Be Processed

YANG files of a device may include deviation statements to define any features not supported by the device, or to describe how device implementation differs from the standard YANG model implementation. In some cases, the generator does not apply the changes specified in the deviation statements. This mismatch causes errors during runtime.

Even if the deviation removes support for a particular data path, the generator still creates Java classes for those nodes. This mismatch causes errors during runtime.

For example, if a schema correctly excludes the "Neighbor" node (as defined in the deviation statement), but the binding generator still creates a class for it, you will see an error like the following:

```
<Failure in Processor Restconf Logical Data Modeller of Action DiscoverRestconfDevices
(Exception: ProcessorException, Base Exception: IncorrectNestingException)>
<Jul 1, 2025, 9:58:02,354 AM Greenwich Mean Time>
<Error>
<oracle.communications.integrity.scanCartridges.sdk.BaseController> <BEA-000000>
```

```
<Action failure: Node
DefaultContainerRuntimeType{javaType=org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.
yang.ietf.ip.rev140616.interfaces._interface.Ipv4,
statement=EmptyContainerEffectiveStatement{argument=(urn:ietf:params:xml:ns:yang:ietf-ip?
revision=2014-06-16)ipv4}} does not have child named interface
org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.ip.rev140616.interfaces._in
terface.ipv4.Neighbor>
```

This error is caused because the schema generator honors the YANG file's deviation statement, but the binding generator does not. This inconsistency leads to conflicts when the generated classes have the fields that the deviation statement has disabled but the schema has not.

Fix: To fix this issue, you must delete the deviation statement for the particular node and re-generate the bindings JAR file.

To apply the fix for the example:

1. Open the relevant deviation YANG file, which is **cisco-xe-ietf-ip-deviation.yang** in this example.
2. Locate the deviation statement for the node that is being disabled or modified which is:

   ```
   deviation /if:interfaces/if:interface/ip:ipv6/ip:neighbor {
   deviate not-supported;
   description "Not supported in IOS-XE 3.17 release.";
   }
   ```

3. Delete the deviation statement.
4. Re-generate the binding JAR files.
5. Use the regenerated JAR file in the NETCONF Network Discovery and UIM Integration cartridge.

Issues Occuring During Binding Generation

This topic describes the issues occurring during the cartridge runtime and their fixes, as listed:

- Class and Package Names Are Case-Sensitive and Do Not Work in Microsoft Windows
- Errors Resulting from Unsupported Use of XPATH 2.0

Class and Package Names Are Case-Sensitive and Do Not Work in Microsoft Windows

The generator sometimes creates classes for containers that have the same fully qualified name but use different cases (for example, *ContainerA* and *containera*). On Windows, file and folder names are treated as case-insensitive which leads to conflicts. As a result, some generated classes overwrite others, causing compilation errors when building the JAR file.

When this happens, you will see an error message like the following when compiling the JAR file:

```
Compilation fails during binding JAT files generation due to unable to
resolve Type:
      ClassName
```

Fix: To fix this error:

1. Navigate to the parent folder where this error is observed.
2. Remove all the generated artifacts.
3. Run this command as an administrator in Windows:

   ```
   fsutil.exe file setCaseSensitiveInfo path enable
   ```

   where *path* is the path of the parent folder.
4. Re-generate the binding JAR files.
5. Use the regenerated JAR file in the NETCONF Network Discovery and UIM Integration cartridge.

Errors Resulting from Unsupported Use of XPATH 2.0

Some YANG files might use XPATH 2.0 which causes syntax errors.

Syntax Error: compare() function

Note

This issue may occur during binding generation and cartridge runtime.

Use of the *compare()* function can generate an error like the following:

```
org.opendaylight.yangtools.yang.parser.spi.source.SourceException: Argument
"not (/ios:native/ios:router/ios-ospf:ospf/ios-ospf:vrf) or compare(/
ios:native/ios:vrf/ios:definition/ios:name, /ios:native/ios:router/ios-
ospf:ospf/ios-ospf:vrf) = 0 and /ios:native/ios:vrf/ios:definition/ios:rd" is
not valid XPath string [at C:\cisco xe
      1691\Cisco-IOS-XE-ospf.yang:1999:7]
```

Fix: To fix this error, use "=" instead of the *compare()* function and ensure that each expression is enclosed in parentheses.

To fix this error:

1. Open the YANG file mentioned in the error, which in this example is **Cisco-IOS-XE-ospf.yang**. The **1999:7** indicates the line number and column number where the syntax error occurs.
2. Rewrite the argument by using the equals sign, for example:

   ```
   (/ios:native/ios:vrf/ios:definition/ios:name = /ios:native/ios:router/ios-ospf:ospf/
   ios-ospf:vrf)
   ```

   Ensure that the revised argument is enclosed in parentheses as shown.
3. Re-generate the binding JAR files.
4. Use the regenerated JAR file in the NETCONF Network Discovery and UIM Integration cartridge.

Syntax Error: Use of Square Brackets

XPATH 2.0 allows the use of square brackets ([ ]). This can generate an error when used in YANG files, like the following:

```
org.opendaylight.yangtools.yang.parser.spi.source.SourceException: Argument "(count(..
[pce]) = 1)" is not valid XPath string [at C:\cisco xe
    1691\Cisco-IOS-XE-mpls.yang:698:7]
```

Fix: To fix this syntax error, rewrite the container name without square brackets preceded by a forward slash.

In the example error above, you can rewrite

```
when "(count(..[pce]) = 1)";
```

as

```
when "(count(../pce) = 1)";
```

Once the syntax error is resolved, regenerate the binding JAR files and use them in the NETCONF Network Discovery and UIM Integration cartridge.